The Surprising Power of Constant Depth Algebraic Proofs

Russell Impagliazzo
Department of Computer Science and Engineering
UC San Diego
russell@cs.ucsd.edu

Sasank Mouli
Department of Electrical and Computer Engineering
UC San Diego
sasankm@ucsd.edu

Toniann Pitassi
Department of Computer Science
University of Toronto
toni@cs.toronto.edu

Abstract

A major open problem in proof complexity is to prove superpolynomial lower bounds for $\mathsf{AC}^0[p]$-Frege proofs. This system is the analog of $\mathsf{AC}^0[p]$, the class of bounded depth circuits with prime modular counting gates. Despite strong lower bounds for this class dating back thirty years ([28, 30]), there are no significant lower bounds for $\mathsf{AC}^0[p]$-Frege. Significant and extensive *degree* lower bounds have been obtained for a variety of subsystems of $\mathsf{AC}^0[p]$-Frege, including Nullstellensatz ([3]), Polynomial Calculus ([9]), and SOS ([14]). However to date there has been no progress on $\mathsf{AC}^0[p]$-Frege lower bounds.

In this paper we study constant-depth extensions of the Polynomial Calculus [13]. We show that these extensions are much more powerful than was previously known. Our main result is that small depth (≤ 43) Polynomial Calculus (over a sufficiently large field) can polynomially effectively simulate all of the well-studied semialgebraic proof systems: Cutting Planes, Sherali-Adams, Sum-of-Squares (SOS), and Positivstellensatz Calculus (Dynamic SOS). Additionally, they can also quasi-polynomially effectively simulate $\mathsf{AC}^0[q]$-Frege for *any* prime q independent of the characteristic of the underlying field. They can also effectively simulate TC^0-Frege if the depth is allowed to grow proportionally. Thus, proving strong lower bounds for constant-depth extensions of Polynomial Calculus would not only give lower bounds for $\mathsf{AC}^0[p]$-Frege, but also for systems as strong as TC^0-Frege.

CCS Concepts: • **Theory of computation → Proof theory**; **Proof complexity**; *Complexity theory and logic.*

Keywords: Proof Complexity, Polynomial Calculus, Algebraic proofs, $\mathsf{AC}^0[p]$-Frege, bounded depth

LICS '20, July 8–11, 2020, Saarbrücken, Germany

ACM ISBN 978-1-4503-7104-9/20/07…$15.00
https://doi.org/10.1145/3373718.3394754

ACM Reference Format:
Russell Impagliazzo, Sasank Mouli, and Toniann Pitassi . 2020. The Surprising Power of Constant Depth Algebraic Proofs. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '20), July 8–11, 2020, Saarbrücken, Germany.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3373718.3394754

1 Introduction

Proof complexity has evolved in parallel to circuit complexity, typically with circuit lower bound techniques being eventually used to show lower bounds for analogous proof systems. One stubborn exception is the analogous proof system for $\mathsf{AC}^0[p]$, the class of bounded depth circuits with prime modular counting gates. Despite strong lower bounds for this class dating back thirty years ([28, 30]), there are no significant lower bounds for $\mathsf{AC}^0[p]$-Frege. Since the only lower bounds for circuits with modular operations are via representations of functions by polynomials ([28, 30]), it seems natural to use algebraic proof systems (e.g, Nullstellensatz ([3]), Polynomial Calculus (PC) ([9]), Positivstellensatz aka Sum-of-Squares (SOS) ([14]), ideal proofs ([15])) to extend these bounds to the proof complexity case. However, despite progress on these proof systems, a super-polynomial lower bound for $\mathsf{AC}^0[p]$-Frege remains open. This paper offers one explanation for this failure: small modifications of these algebraic proof systems to handle constant depth overshoot and allow reasoning far beyond that possible by $\mathsf{AC}^0[p]$ circuits.

Since lower bounds for Polynomial Calculus itself do not imply lower bounds for $\mathsf{AC}^0[p]$-Frege systems, various researchers have suggested ways to strengthen PC to create algebraic systems which do p-simulate $\mathsf{AC}^0[p]$-Frege ([8, 13, 23]). Unfortunately, it is not clear how to extend lower bound techniques for PC to these systems. As an illustration of how small extensions can increase the power of these proof systems, consider Polynomial Calculus where we allow changes of bases. Many strong lower bounds are known for the size of PC proofs for tautologies like the Pigeonhole Principle [29], [19] and Tseitin tautologies [5]. All of the above lower bounds use a degree-size connection, which roughly states that a linear lower bound on the degree of any refutation translates to an exponential lower bound on its size. But this connection is highly basis dependent. The connection only holds true over the $\{0, 1\}$ basis, and even allowing a

change to the $\{-1, 1\}$ basis immediately gives a polynomial sized proof for the *mod* 2 Tseitin tautologies. Grigoriev and Hirsch [13] noted the above and in addition showed that allowing for introduction of new variables which are linear transformations of the original variables gives a short proof of the Pigeonhole principle as well. They also generalized the notion of a linear transformation by considering transformations obtained by applying constant depth arithmetic circuits and arithmetic formulas to the original variables. The resulting systems turn out to be quite powerful, and it is shown in [13] that the latter simulates Frege systems, and the former simulates depth d $\mathsf{AC}^0[\mathsf{p}]$-Frege proofs by using arithmetic circuits of depth $d' = \Theta(d)$. Raz and Tzameret [27] defined a proof system along similar lines where the transformations are restricted such that each line of the proof is a multilinear formula in the original variables. It was shown that even under these restrictions, linear transformations allow small proofs of the functional Pigeonhole principle and Tseitin tautologies. They also showed in [26] that Polynomial Calculus with added linear transformations simulates the system $R(CP^*)$ of Krajicek [20], which is stronger than Cutting Planes with bounded coefficients.

1.1 Our Work

Here, we show that these extensions to PC are even more powerful than previously known. Over a sufficiently large field of characteristic p, the same extensions that allow PC to simulate depth d $\mathsf{AC}^0[\mathsf{p}]$ proofs also allows it to simulate much stronger proof systems. So to prove a lower bound on $\mathsf{AC}^0[\mathsf{p}]$ proofs via such systems would seem to require proving lower bounds for systems as strong as TC^0-Frege.

More precisely, consider the following additions to PC. In an additive extension, we introduce a new variable y and a new defining equation $y = \sum a_i x_i + b$ where $a_i, b \in \mathbb{F}$. In a multiplicative extension, we introduce a new variable y and a new defining equation $y = b \prod (x_i)^{e_i}$. Depth-d-PC allows the usual (syntactic) reasoning of Polynomial Calculus using these extension variables (i.e. multiplying a line by the variable y is allowed), with each line having up to $d - 2$ alternating layers of additive and multiplicative extensions. (The new variables in a depth d-PC proof are equivalent to depth $d - 2$ algebraic circuits, and polynomials in terms of these variables are depth d algebraic circuits.)

A note regarding the notion of simulation. All our simulation results below use the notion of *effective simulation* from [25] (see Definition 4). For the rest of the paper, "simulate" refers to an effective simulation. This is an important distinction, since Alekseev et. al. [1] show that assuming the Shub-Smale hypothesis, even very strong algebraic systems like the Ideal Proof System [16] cannot simulate (in the usual sense – see Definition 3) weak semi-algebraic systems like SOS.

We remove the restriction of polynomially bounded coefficients from the result of [26] and show how to perform arithmetic with large coefficients, and as a result effectively simulate Cutting Planes with unbounded coefficients and the Sum-of-Squares (SOS) proof system. (Our theorem works for the stronger system Positivstellensatz Calculus [14]).

Theorem 1. *Depth-43-PC can effectively* p*-simulate Cutting Planes and Positivstellensatz Calculus over $\mathbb{F}_{p^m}$ for any prime p, where m is logarithmic in the maximum number of monomials in any proof line.*

Clote and Kranakis [10] mention a proof, due to Krajíček, of Cutting Planes being simulated by the bounded-depth threshold logic system PTK of Buss and Clote [7]. Since we simulate a modified version of PTK to show Theorem 2 below, it already follows that our system simulates Cutting Planes. However, the above proof by Krajíček is non-explicit and does not provide a value of the depth at which the simulation happens. Determining this value is posed as an open problem in [10]. Theorem 1 provides an upper bound of $d \leq 43$ through an explicit simulation. The following is a brief outline. Given a polynomial P, to assert that it is non-negative using a set of low depth polynomial equations, we introduce a signed bit representation for P by representing the coefficient of each monomial using the bit vector of its two's complement representation, and performing bitwise addition over these bit vectors to obtain a bit vector representing P. The method of carrying out bitwise addition is carefully chosen so that not only is the resultant vector of low depth, but the correctness of this process can also be proved in low depth, which is crucial to carrying out the simulation. With this representation in hand, the line $P \geq 0$ is represented by $sign(P) = 0$ where *sign* indicates the sign bit (which is zero if and only if the number being represented is non-negative). The simulation of the semi-algebraic proof system is then carried out step by step, where a derivation of $P_2 \geq 0$ from $P_1 \geq 0$ is mimicked by deriving $sign(P_2) = 0$ from $sign(P_1) = 0$. Finally, since a semi-algebraic refutation can be assumed to end with the line $-1 \geq 0$, our simulation gives us the line $sign(-1) = 0$, which is simply the line $1 = 0$, a contradiction. The constant of 43 is obtained since the construction we use for low depth bitwise addition is of depth about 10, but proving its correctness requires stacking up four layers of it. We have not tried to optimize this constant, and this is just a rough estimate. Theorem 1 is proved in section 4.4.

We improve the results of Grigoriev and Hirsch in the constant depth case in two ways. We show that $\mathsf{AC}^0[\mathsf{p}]$-Frege can be simulated with a fixed constant depth, but with a quasipolynomial blowup. Significantly, this simulation also simulates modular gates of different characteristic than the field we are working over.

Theorem 2. *Let p be an arbitrary prime and n be a positive integer. For some $m = O(\mathsf{poly}(\log(n)))$, depth-9-PC over $\mathbb{F}_{p^m}$*

can effectively quasipolynomially simulate $\mathsf{AC}^0[q]$*-Frege over n variables for any prime q.*

Buss *et al.* [6] showed that an $\mathsf{AC}^0[p]$-Frege proof of depth d can be collapsed to a depth 3 $\mathsf{AC}^0[p]$-Frege proof with a quasipolynomial blowup. In conjunction with [13], this implies the above theorem for the case of $q = p$. Thus, apart from being more general, our result also provides an alternative and perhaps simpler proof of the case of $q = p$. We prove Theorem 2 in sections 4.2.1 and 4.2.2.

We also show that allowing for arbitrarily large but constant depth transformations enables the simulation of TC^0-Frege.

Theorem 3. *A* TC^0*-Frege proof of depth d can be effectively* p*-simulated by depth-d′-PC over* $\mathbb{F}_{p^m}$*, where* $d' = O(d)$ *and m is logarithmic in the size of the largest threshold gate, for any prime p.*

The proof of Theorem 3 is shown in section 4.3.

We also improve the results of Raz and Tzameret [26] to show that Polynomial Calculus with linear transformations can simulate *semantic* Cutting Planes with small coefficients.

Theorem 4. *Depth-3-PC can effectively* p*-simulate semantic CP* over* $\mathbb{Q}$*.*

Theorem 4 is proved in sections 3.1 and 3.2.

1.2 Related Work

Pitassi [23, 24] introduced powerful generalizations of the Polynomial Calculus that operate directly on formulas. Grochow and Pitassi [16] introduced the more general IPS proof system, and proved that superpolynomial lower bounds for IPS would imply the longstanding problem of separating VP from VNP. However, these algebraic systems are not Cook-Reckhow proof systems since proofs are not known to be checkable in polynomial time (but rather in randomized polynomial-time.)

In 2003, Grigoriev and Hirsch [13] introduced a Cook-Reckhow style algebraic proof system for formulas, with derivation rules corresponding to the ring axioms. Motivated by understanding how many basic ring identities are needed to verify polynomial identities, Hrubes and Tzameret [17] introduced a very closely related equational proof system for proving polynomial identities over a ring. Even earlier, [8] study essentially the same proof system but where the focus is over finite fields. Finally, Raz and Tzameret [26] introduced the $Res(lin)$ proof system, which generalizes Resolution using extension variables given by linear forms, in a similar way to our generalization of PC using extension variables. They also showed that $Res(lin)$ simulates the system $R(CP^*)$ (defined in [20]) and Polynomial Calculus over depth 3 formulas can simulate $Res(lin)$. Alekseev et. al. [1] also considered generalized versions of Nullstellensatz and Sum-of-Squares over algebraic circuits of arbitrary depth. Conditioned on the assumption that a certain subset sum principle has a small IPS proof, they make use of bitwise arithmetic to show that these systems are equivalent to IPS. Although we also use bitwise arithmetic to prove Theorem 1, our work vastly differs from theirs in the following aspects. Firstly, the proof systems considered by them are not Cook-Reckhow systems, i.e. it is not known whether the proofs in these systems can be verified in deterministic polynomial time. These systems are hence much more powerful than the ones we consider here, and in particular they are not concerned with performing bitwise arithmetic in constant depth, which is the main focus of our simulations. Secondly, while we use the notion of *effectively p-simulation* [25] for all our results, they chiefly focus on the more conventional notion of p-simulation. Effective simulation allows for a formula in the simulated system to be "pre-processed" in a truth-preserving way before it is represented in the simulating system, while p-simulation is only defined for two proof systems which can express the same set of formulae.

1.3 Organization of the paper

The rest of the paper is organized as follows. In section 2.1, we discuss some basic definitions and notations. In section 2.2, we define the notions from proof complexity and proof systems used in this paper. In section 2.3, we formalize the system of bounded depth Polynomial Calculus. In section 3.1, we sketch the simulation of syntactic Cutting Planes with bounded coefficients from [26], since it is essential for a significant part of the subsequent discussion. In section 3.2, we extend the simulation to the semantic case, proving Theorem 4. In section 4.1, we prove an analog of the results in section 3.1 over a large enough finite field extension, for use in subsequent sections. In sections 4.2.1, 4.2.2, 4.3, we use techniques from this analog to prove Theorems 2 and 3. Finally in section 4.4, we prove Theorem 1. Technical details of simulations from each of the above sections are contained in the full version [18].

2 Preliminaries and Generalizations of Polynomial Calculus

2.1 Preliminaries

2.1.1 Notation. Integers are represented by letters a, b, c. For an integer a, let $a^+ = a$ if $a > 0$ and 0 otherwise. Define $|a|$ to be the length in binary of a. Sets of integers are represented by letters A, B, C. Indices to sets are represented by letters i, j, k, ℓ.

Variables are represented by x, y, z, w where x usually represents the original variables and the others represent the extension variables. Monomials are represented by upper case letters X, Y, Z. Polynomials are represented by P, Q, R. Boolean formulae are represented by φ.

We treat all the above as one dimensional objects. Multi-dimensional objects, or vectors, are represented in boldface.

Constant vectors are represented by $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$. Vectors whose components may be variables or polynomials are represented by $\mathbf{y}$, $\mathbf{z}$, $\mathbf{w}$.

Calligraphic letters $\mathcal{R}$, $\mathcal{S}$ are used for special expressions which are contextual.

Definition 1 (Straight Line Program (SLP)). *A SLP S over variables $\{x_1, \ldots, x_n\}$ and a field $\mathbb{F}$ is a sequence of computations $(y_1, \ldots, y_k)$ such that each y_j is equal to one of the following, where $C_j \subseteq \{1, \ldots, j-1\}$*

x_i *for some* $i \in \{1 \cdots n\}$
$\sum_{\ell \in C_j} \alpha_\ell y_\ell$ *for some constants* $\alpha_\ell \in \mathbb{F}$
$\prod_{\ell \in C_j} y_\ell$

We view a SLP as a directed acyclic graph where internal nodes are labelled with either Product or Plus gates and the leaf nodes are labelled with a variable x_i. The size of a SLP is therefore the number of nodes in the corresponding directed acyclic graph, and the depth is the maximum number of nodes on a root to leaf path in the directed acyclic graph.

2.2 Propositional proof systems

Definition 2 (Cook-Reckhow proof system). *For a language $L \subseteq \{0,1\}^*$, a Cook-Reckhow proof system is a polynomial time deterministic verifier V such that*

- *If $x \in L$, there exists a proof π such that $V(x, \pi)$ accepts.*
- *If $x \notin L$, for all proofs π, $V(x, \pi)$ rejects.*

Definition 3 (p-simulation). *For two proof systems V_1 and V_2 defined over the same language L, V_2 is said to p-simulate V_1 if there exists a polynomial time computable function f such that for every $x \in L$, if π_1 is a proof of x for V_1, $f(\pi_1)$ is a proof of x for V_2.*

Definition 4 (Effectively p-simulation [25]). *For two proof systems V_1 and V_2 over languages L_1 and L_2, V_2 is said to effectively p-simulate V_1 if there exist polynomial time computable functions f, g such that $x_1 \in L_1$ if and only if $g(x_1) \in L_2$ and if π_1 is a proof of x_1 for V_1, $f(\pi_1)$ is a proof of $g(x_1)$ for V_2.*

In this paper, we are only concerned with effective simulations. The propositional proof systems we will work with are defined below.

Definition 5 (Cutting Planes). *Let $\Delta = \{A_1, \ldots, A_m\}$ be a set of unsatisfiable integer linear inequalities in boolean variables $x_1, \ldots, x_n$ of the form $A_j \equiv \sum_i a_{ij} x_i \geq b_i$ where a_{ij} and b_i are integers. A Cutting Planes refutation of Δ is a sequence of lines $B_1, \ldots, B_s$ such that B_s is the inequality $0 \geq 1$ and for every $\ell \in \{1, \ldots, s\}$ $B_\ell \in \Delta$ or is obtained through one of the following derivation rules for $j, k < \ell$*

Addition. *From $B_j \equiv \sum_i c_{ij} x_i \geq d_j$ and $B_k \equiv \sum_i c_{ik} x_i \geq d_k$, derive*

$$\sum_i (c_{ij} + c_{ik}) x_i \geq d_j + d_k$$

Multiplication by a constant. *From $B_j \equiv \sum_i c_{ij} x_i \geq d_j$, derive*

$$c \sum_i c_{ij} x_i \geq c d_j$$

for an integer $c \geq 0$.

Division by a nonzero constant. *From $B_j \equiv \sum_i c_{ij} x_i \geq d_j$ and an integer $c > 0$ such that c divides c_{ij} for all i, derive*

$$\sum_i \frac{c_{ij}}{c} x_i \geq \lceil d_j / c \rceil$$

The semantic version of the system also has the following rule

Semantic inference. *If $B_j \equiv \sum_i c_{ij} x_i \geq d_j$, $B_k \equiv \sum_i c_{ik} x_i \geq d_j$ and $B_\ell \equiv \sum_i c_{i\ell} x_i \geq d_j$ are inequalities such that every assignment to $x_1, \ldots, x_n$ that satisfies B_j and B_k also satisfies B_ℓ, then from lines B_j and B_k, derive B_ℓ.*

The size of a line is the size of its bit representation. The size of a proof is the sum of sizes of each line. The length of a Cutting Planes proof is equal to the number of lines in the proof. We define the coefficient size of a Cutting Planes proof to be equal to the maximum of the absolute values of all the constants that appear in the proof. $\mathbf{CP}^*$ is a subsystem of Cutting Planes where the coefficient size is bounded by a polynomial in the number of variables. Without loss of generality, the coefficient size can be bounded by $2^{\mathrm{poly}(\ell)}$ where ℓ is the length of the proof due to [11].

Definition 6 (Polynomial Calculus (PC)). *Let $\Gamma = \{P_1, \ldots, P_m\}$ be a set of polynomials in variables $\{x_1, \ldots, x_n\}$ over a field $\mathbb{F}$ such that the system of equations $P_1 = 0, \ldots, P_m = 0$ has no solution. A Polynomial Calculus refutation of Γ is a sequence of polynomials $R_1, \ldots, R_s$ where $R_s = 1$ and for every ℓ in $\{1, \ldots, s\}$, $R_\ell \in \Gamma$ or is obtained through one of the following derivation rules for $j, k < \ell$*

$R_\ell = \alpha R_j + \beta R_k$ *for* $\alpha, \beta \in \mathbb{F}$
$R_\ell = x_i R_k$ *for some* $i \in \{1, \ldots, n\}$

The size of the refutation is $\sum_{\ell=1}^{s} |R_\ell|$, where $|R_\ell|$ is the number of monomials in the polynomial R_ℓ. The degree of the refutation is $\max_\ell deg(R_\ell)$.

The following system is known to simulate PC, SOS and Sherali-Adams.

Definition 7 (Positivstellensatz Calculus/Dynamic SOS [14]). *Let $\Gamma = \{P_1, \ldots, P_m\}$ and $\Delta = \{Q_1, \ldots, Q_r\}$ be two sets of polynomials over $\mathbb{R}$ such that the system of equations $P_1 = 0, \cdots, P_m = 0$, $Q_1 \geq 0, \cdots, Q_r \geq 0$ is unsatisfiable. A Dynamic SOS refutation of Γ, Δ is a sequence of inequalities $R_1 \geq 0, \ldots, R_s \geq 0$ where $R_s = -1$ and for every ℓ in $\{1, \ldots, s\}$, $R_\ell \in \Gamma \cup \Delta$ or is obtained through one of the following derivation rules for $j, k < \ell$*

1. *From $R_j = 0$ and $R_k = 0$ derive $\alpha R_j + \beta R_k = 0$ for $\alpha, \beta \in \mathbb{R}$*
2. *From $R_k = 0$ derive $x_i R_k = 0$ for some $i \in \{1, \ldots, n\}$*

3. *From $R_j \geq 0$ and $R_k \geq 0$ derive $\alpha R_j + \beta R_k \geq 0$ for $\alpha \geq 0$, $\beta \geq 0 \in \mathbb{R}$*
4. *From $R_j \geq 0$ and $R_k \geq 0$ derive $R_j R_k \geq 0$*
5. *Derive $R^2 \geq 0$ for some polynomial $R \in \mathbb{R}[x_1, \ldots, x_n]$*

The size of a line is the size of its bit representation. The size of a Dynamic SOS refutation is the sum of sizes of each line of the refutation.

2.3 Generalizations of Polynomial Calculus

We now define a variant of Polynomial Calculus, ΣΠΣ-PC where the proof system is additionally allowed to introduce new variables y_j corresponding to affine forms in the original variables x_i. Thus, each line of the proof is represented by a ΣΠΣ algebraic circuit.

Definition 8 (ΣΠΣ-PC)**.** *Let $\Gamma = \{P_1, \ldots, P_m\}$ be a set of polynomials in variables $\{x_1, \ldots, x_n\}$ over a field $\mathbb{F}$ such that the system of equations $P_1 = 0, \ldots, P_m = 0$ has no solution. A ΣΠΣ-PC refutation of Γ is a Polynomial Calculus refutation of a set $\Gamma' = \{P_1, \ldots, P_m, Q_1, \ldots, Q_k\}$ of polynomials over variables $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_k\}$ where $Q_1, \ldots, Q_k$ are polynomials of the form $Q_j = y_j - (a_{j0} + \sum_i a_{ij} x_i)$ for some constants $a_{ij} \in \mathbb{F}$.*

The size of a ΣΠΣ-PC refutation is equal to the size of the Polynomial Calculus refutation of Γ'.

We would now like to generalize the above proof system to an arbitrary depth d.

Definition 9 (Depth-d-PC)**.** *Let $d > 2$ be an integer. Let $\Gamma = \{P_1, \ldots, P_m\}$ be a set of polynomials in variables $\{x_1, \ldots, x_n\}$ over a field $\mathbb{F}$ such that the system of equations $P_1 = 0, \ldots, P_m = 0$ has no solution. Let $S = (y_1, \ldots, y_k)$ be a SLP over $\{x_1, \ldots, x_n\}$ and $\mathbb{F}$ of depth $d-2$ defined by $y_j = Q_j(x_1, \ldots, x_n, y_1, \ldots, y_{j-1})$. A depth-$d$-PC refutation of Γ is a Polynomial Calculus refutation of the set $\Gamma' = \{P_1, \ldots, P_m, y_1 - Q_1, \ldots, y_k - Q_k\}$ of polynomials over $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_k\}$.*

The size of a depth-d-PC refutation is the size of the Polynomial Calculus refutation of Γ'

Viewing a refutation in depth-d-PC as a depth d algebraic circuit in the original variables $\{x_1, \ldots, x_n\}$ (with each line of the refutation being a gate in the circuit), it is easy to see that the above definition of size for a refutation coincides with the usual notion of size for an algebraic circuit up to polynomial factors.

Although we define the size of a proof in depth-d-PC in terms of the number of monomials, we will be using the number of lines as a measure of the size, since in our simulations no line contains more than a polynomial number of monomials.

To conclude this section, we state the following result from [26], which is the starting point of our work.

Theorem 0. ***[26]*** *ΣΠΣ-PC over $\mathbb{Q}$ can simulate syntactic Cutting Planes with size polynomial in n and the coefficient size.*

3 Simulations over $\mathbb{Q}$

In this section we outline how we translate inequalities into polynomials over $\mathbb{Q}$, and simulate proofs involving these inequalities into Polynomial Calculus derivations over their translations.

Consider a line $A_j \equiv \sum_i a_{ij} x_i \geq b_j$ in a CP* proof, where $|a_i|$, $|b|$ are bounded logarithmically in n. We define its translation over $\mathbb{Q}$ as the following

Definition 10 (Translation from CP* to ΣΠΣ-PC)**.** *For a line $A_j \equiv \sum_i a_{ij} x_i \geq b_j$ its translation in ΣΠΣ-PC is defined to be the following pair of lines*

$$\prod_{b=0}^{\sum_i a_{ij}^+ - b_j} (y_j - b) = 0$$

$$y_j = \sum_i a_{ij} x_i - b_j$$

In addition, for all i, the equations $x_i(x_i - 1) = 0$ are included in the translation.

That is, we introduce a variable $y_j = \sum_i a_{ij} x_i - b_j$ and indicate the range of values it can take which satisfy the constraint $\sum_i a_{ij} x_i \geq b_j$. For convenience, we will denote by $z \in A$ the equation $\prod_{a \in A}(z - a) = 0$.

The key idea is to note that given two equations $z \in A$ and $z \in B$, we can derive in ΣΠΣ-PC the equation $z \in A \cap B$. We call this the Intersection lemma. A formal proof is provided in the full version [18].

3.1 Simulating syntactic CP*

We now sketch how all the derivations rules of syntactic CP* can be simulated with the help of the Intersection lemma, concluding Theorem 0 (originally proved in [26]). For instance, given equations $y_1 \in A$ and $y_2 \in B$, we derive the range of values a variable $z = y_1 + y_2$ takes as follows. For every $a_1 \in A$, we derive an equation which states $z \in a_1 + B$ OR $y_1 \in A \setminus \{a_1\}$ where $a_1 + B = \{a_1 + b \mid b \in B\}$. This equation is formally represented as

$$\prod_{c \in a_1 + B} (z - c) \prod_{a \in A \setminus \{a_1\}} (y_1 - a) = 0$$

We can multiply each of these equations by appropriate variables, so that the part about z is the same in all of them. We would now like to eliminate the part about y_1 from these equations. Noting that $\cap_i A \setminus \{a_i\} = \emptyset$, we use the Intersection lemma inductively to eliminate y_1.

For simulating division by an integer c given a variable $z = \sum_i c_i x_i$ and an equation $z \in C$ such that c divides every element of C, we first derive $z \in I$, where I is all possible

integer values of the expression $\sum_i c_i x_i$, by using our simulation of addition. We then introduce a variable $z' = z/c$ and from the former equation, we get a set of integer values for z' and from the latter, we get a set of rational values. Using the Intersection lemma now gives the right range for the variable $z' = z/c$.

For a formal proof, see the full version of the paper [18].

3.2 Simulating semantic CP*

In this section we extend the above simulation to include semantic CP*, hence completing the proof of Theorem 4. Let $L_1 \equiv \sum_i a_i x_i \geq d_1$, $L_2 \equiv \sum_i b_i x_i \geq d_2$ be two lines in a Cutting Planes proof and let $L_3 \equiv \sum_i c_i x_i \geq d_3$ be a semantic consequence of L_1 and L_2. Let $y = \sum_i a_i x_i$, $z = \sum_i b_i x_i$ and $w = \sum_i c_i x_i$. Let $A = \{0, \ldots, \sum_i a_i^+\}$, $B = \{0, \ldots, \sum_i b_i^+\}$ and $C = \{0, \ldots, \sum_i c_i^+\}$. Using the simulation of addition in syntactic CP*, we can derive the equations

$$\prod_{a \in A}(y - a) = 0$$
$$\prod_{b \in B}(z - b) = 0$$
$$\prod_{c \in C}(w - c) = 0$$

This restricts the values that can be taken by the tuple (y, z, w) to the three dimensional grid $A \times B \times C$. Let a point (i, j, k) in the grid be *infeasible* if the tuple (y, z, w) never evaluates to it for any assignment to $\{x_i\}$. Our first step is to derive *infeasibility equations* of the form

$$\prod_{\substack{a \in A \\ a \neq i}}(y - a) \prod_{\substack{b \in B \\ b \neq j}}(z - b) \prod_{\substack{c \in C \\ c \neq k}}(w - c) = 0$$

which for $(i, j, k) \in A \times B \times C$ tells us that the point (i, j, k) in the grid is infeasible for the tuple (y, z, w).

Lemma 1. *For every infeasible point $(i, j, k) \in A \times B \times C$, $\Sigma\Pi\Sigma$-PC can derive an infeasibility equation of the above form in $O((\sum_i a_i^+)^2 (\sum_i b_i^+)^2 (\sum_i c_i^+)^2)$ lines*

The proof of this lemma is left to the full version [18].

The next step is to use the ranges of y and z specified in lines L_1 and L_2 to narrow down the possible values that can be taken by w. Our goal will be to get an equation of the form

$$\prod_{c \in C'}(w - c) = 0$$

such that each c in C' is feasible for w under the constraints L_1 and L_2 on y and z respectively.

Let P_i be the translation of L_i in $\Sigma\Pi\Sigma$-PC, for $i = 1, 2, 3$. Let $\mathcal{I}_{a,b}$ denote the set of all infeasibility equations for points of the form (a, b, k) for some $k \in C$. For an equation P of the form $\prod_{a \in A_1}(y - a) \prod_{b \in B_1}(z - a) \prod_{c \in C_1}(w - a) = 0$, denote by $\mathcal{R}_y(P)$ the set A_1, that is the range of values specified by the equation for the variable y. $\mathcal{R}_z$ and $\mathcal{R}_w$ are defined analogously. We describe how to obtain the set C' by the algorithm w-FEASIBLE which operates on the range sets.

```
procedure w-FEASIBLE(P1,P2)
    C' ← ∅
    for (a, b) ∈ R_y(P1) × R_z(P2) do
        S ← C
        for I ∈ I_{a,b} do
            S ← S ∩ R_w(I)
        end for
        C' ← C' ∪ S
    end for
    return C'
end procedure
```

Consider a pair $(a, b) \in \mathcal{R}_y(P_1) \times \mathcal{R}_z(P_2)$. For any equation $I \in \mathcal{I}_{a,b}$, $\mathcal{R}_w(I)$ gives a list of possible values the variable w can take when $(y, z) = (a, b)$. By Lemma 1, $(y, z, w) = (a, b, c)$ is infeasible if and only if there is an equation $I \in \mathcal{I}_{a,b}$ such that $c \notin \mathcal{R}_w(I)$. Therefore, $\bigcap_{I \in \mathcal{I}_{a,b}} \mathcal{R}_w(I)$ is precisely the feasible set of values for w, given $(y, z) = (a, b)$. C' is the union of such sets over all possible pairs $(a, b) \in \mathcal{R}_y(P_1) \times \mathcal{R}_z(P_2)$ and hence is the set of all feasible values of w.

This algorithm over range sets can be easily translated to a proof of $\prod_{c \in C'}(w - c) = 0$ from P_1 and P_2 in $\Sigma\Pi\Sigma$-PC as follows. To simulate the inner **for** loop, we use the Intersection lemma inductively over all equations in $\mathcal{I}_{a,b}$ to get equations $J_{a,b}$ such that $\mathcal{R}_w(J_{a,b}) = \bigcap_{I \in \mathcal{I}_{a,b}} \mathcal{R}_w(I)$. Note that $\mathcal{R}_y(J_{a,b}) = A \setminus \{a\}$ and $\mathcal{R}_z(J_{a,b}) = B \setminus \{b\}$. Thus using the Intersection lemma again inductively over the set $\{J_{a,b}\}$ (analogous to simulation of addition in syntactic CP*) would give an equation free of y and z, where w ranges over $\bigcup_{(a,b)} \mathcal{R}_w(J_{a,b})$. Any semantic consequence P_3 must be such that $\mathcal{R}_w(P_3) \supseteq C'$ and hence is easily derived.

4 Simulations over $\mathbb{F}_{p^m}$

4.1 Simulating syntactic CP*

We now carry out the simulation from [26] in Section 3.1 in depth-d-PC over a large enough field extension F_{p^m} of a finite field F_p. This will be of use in the next section, where we simulate $AC^0[p]$-Frege in depth-d-PC over F_{p^m}. For the following discussion, we set $d = 5$.

To represent large integers over $\mathbb{F}_{p^m}$, we choose a primitive element α and for each of the original variables x_i perform the linear transformation $y_i = 1 + (\alpha - 1)x_i$. Since x_i is boolean, y_i is essentially equivalent to the mapping $x_i \mapsto \alpha^{x_i}$. The expression $\sum_i a_i x_i$ is thus represented as $\alpha^{\sum_i a_i x_i}$. The goal here is to show that all the steps of the simulation in section 3.1 can still be performed after this transformation.

Theorem 5. *Depth-d-PC over F_{p^m} can simulate syntactic Cutting Planes with the number of lines polynomial in n and the*

coefficient size, where m is logarithmic in n and the coefficient size.

Let s_1 be the coefficient size of the Cutting Planes proof. Define $s = ns_1$. Choose m to be the smallest integer such that $2s^2 < p^m - 1$. Let α be an arbitrary primitive element of $\mathbb{F}_{p^m}$.

Definition 11 (Translation of Cutting Planes to depth-d-PC over $\mathbb{F}_{p^m}$). *The translation of $\sum_i a_i x_i \geq b_i$ is defined as follows, where y_i and y are new variables.*

$$y_i = (\alpha^{a_i} - 1)x_i + 1$$

$$y = \prod_i y_i$$

$$(y - \alpha^{b_i})(y - \alpha^{b_i+1}) \cdots (y - \alpha^{\sum_i a_i^+}) = 0$$

An integer c such that $0 \leq c \leq s$ is represented as α^c, whereas for $-s \leq c < 0$ we represent it as $\alpha^{-|c|} \equiv \alpha^{(p^m-1)-|c|}$. Since $2s \leq 2s^2 < p^m - 1$, these representations are unique.

The technical details of the simulating the rules of CP are largely similar to that over $\mathbb{Q}$ and are hence left to the full version [18].

4.2 Simulating $\mathsf{AC}^0[q]$-Frege

4.2.1 Case of $q = p$. For the purpose of this section, we set $d = 9$. We will use the simulation of $\mathsf{AC}^0[p]$-Frege in [21] to show that the same can be carried out in depth-d-PC over $\mathbb{F}_{p^m}$. We fix m to be a large enough integer such that $m = O(\mathsf{poly}(\log(n)))$, so that the field we are working over is quasipolynomial sized. Below we describe the proof system of [21] and their simulation of $\mathsf{AC}^0[p]$-Frege.

The Proof System of Maciel and Pitassi. Maciel and Pitassi [21] define a proof system with mod p, negation, AND, OR and threshold connectives, based on the system PTK by Buss and Clote [7] which we describe below.
Connectives Let $x_1 \cdots x_n$ be boolean variables. For $0 \leq j < p$, let $\oplus_j^p(x_1 \cdots x_n)$ denote the connective which is 1 if and only if $\sum_i x_i = j \mod p$. For any integer t, let $Th_t(x_1 \cdots x_n)$ denote the connective which is 1 if and only if $\sum_i x_i \geq t$. Let $\wedge(x_1 \cdots x_n)$, $\vee(x_1 \cdots x_n)$ denote AND and OR connectives of arity n and $\neg$ denote the NOT gate.

The proof system of Maciel and Pitassi [21]

initial sequents.

1. $\varphi \to \varphi$ for any formula φ
2. $\to \wedge() \,;\, \vee() \to$
3. $\oplus_j^p() \to$ for $1 \leq j < p$; $\to \oplus_0^p()$
4. $Th_t() \to$
5. $\to Th_0(\varphi_1 \cdots \varphi_k)$ for any $k \geq 0$

structural rules.

weakening: $\dfrac{\Gamma, \Delta \to \Gamma'}{\Gamma, \varphi, \Delta \to \Gamma'} \quad \dfrac{\Gamma \to \Gamma', \Delta'}{\Gamma \to \Gamma', \varphi, \Delta'}$

contract: $\dfrac{\Gamma, \varphi, \varphi, \Delta \to \Gamma'}{\Gamma, \varphi, \Delta \to \Gamma'} \quad \dfrac{\Gamma \to \Gamma', \varphi, \varphi, \Delta'}{\Gamma \to \Gamma', \varphi, \Delta'}$

permute: $\dfrac{\Gamma, \varphi_1, \varphi_2, \Delta \to \Gamma'}{\Gamma, \varphi_2, \varphi_1, \Delta \to \Gamma'} \quad \dfrac{\Gamma \to \Gamma', \varphi_1, \varphi_2, \Delta'}{\Gamma \to \Gamma', \varphi_2, \varphi_1, \Delta'}$

cut rule.

$$\frac{\Gamma, \varphi \to \Delta \qquad \Gamma' \to \varphi, \Delta'}{\Gamma, \Gamma' \to \Delta, \Delta'}$$

logical rules.

$\neg$: $\dfrac{\Gamma \to \varphi, \Delta}{\neg\varphi, \Gamma \to \Delta} \quad \dfrac{\varphi, \Gamma \to \Delta}{\Gamma \to \neg\varphi, \Delta}$

$\wedge$-left: $\dfrac{\varphi_1, \wedge(\varphi_2 \cdots \varphi_k), \Gamma \to \Delta}{\wedge(\varphi_1 \cdots \varphi_k), \Gamma \to \Delta}$

$\wedge$-right: $\dfrac{\Gamma \to \varphi_1, \Delta \qquad \Gamma \to \wedge(\varphi_2 \cdots \varphi_k), \Delta}{\Gamma \to \wedge(\varphi_1, \varphi_2 \cdots \varphi_k), \Delta}$

$\vee$-left: $\dfrac{\varphi_1, \Gamma \to \Delta \qquad \vee(\varphi_2 \cdots \varphi_k), \Gamma \to \Delta}{\vee(\varphi_1, \varphi_2 \cdots \varphi_k), \Gamma \to \Delta}$

$\vee$-right: $\dfrac{\Gamma \to \varphi_1, \vee(\varphi_2 \cdots \varphi_k), \Delta}{\Gamma \to \vee(\varphi_1 \cdots \varphi_k), \Delta}$

$\oplus_i$-left:
$$\frac{\varphi_1, \oplus_{i-1}^p(\varphi_2 \cdots \varphi_k), \Gamma \to \Delta \qquad \oplus_i^p(\varphi_2 \cdots \varphi_k), \Gamma \to \varphi_1, \Delta}{\oplus_i^p(\varphi_1, \varphi_2 \cdots \varphi_k), \Gamma \to \Delta}$$

$\oplus_i$-right:
$$\frac{\varphi_1, \Gamma \to \oplus_{i-1}^p(\varphi_2 \cdots \varphi_k), \Delta \qquad \Gamma \to \varphi_1, \oplus_i^p(\varphi_2 \cdots \varphi_k), \Delta}{\Gamma \to \oplus_i^p(\varphi_1, \varphi_2 \cdots \varphi_k), \Delta}$$

Th_t-left:
$$\frac{Th_t(\varphi_2 \cdots \varphi_k), \Gamma \to \Delta \qquad \varphi_1, Th_{t-1}(\varphi_2 \cdots \varphi_k), \Gamma \to \Delta}{Th_t(\varphi_1, \varphi_2 \cdots \varphi_k), \Gamma \to \Delta}$$

Th_t-right:
$$\frac{\Gamma \to \varphi_1, Th_t(\varphi_2 \cdots \varphi_k), \Delta \qquad \Gamma \to Th_{t-1}(\varphi_2 \cdots \varphi_k), \Delta}{\Gamma \to Th_t(\varphi_1, \varphi_2 \cdots \varphi_k), \Delta}$$

Formulas. A *formula* is recursively defined as follows. Input variables $x_1 \cdots x_n$ are formulas of size 1 and depth 1. A formula φ is an expression of the form $g(\varphi_1 \cdots \varphi_k)$, where g

is any of the connectives described above and $\varphi_1 \cdots \varphi_k$ are formulas. The $depth(\varphi)$ is defined as $\sum_{i=1}^{k} depth(\varphi_i)+1$. The $size(\varphi)$ is defined as $\sum_{i=1}^{k} size(\varphi_i)+k+1$ if g is not a threshold connective, and it is defined as $\sum_{i=1}^{k} size(\varphi_i)+t+k+1$ if g is a threshold connective of the form $Th_t(\varphi_1 \cdots \varphi_k)$.
Cedents and Sequents. A cedent Γ is defined as a sequence of formulas $\varphi_1 \cdots \varphi_k$. We will use capital Greek letters to denote cedents. A sequent is an expression of the form $\Gamma \rightarrow \Delta$, where Γ and Δ are cedents. The interpretation of a sequent is that the AND of all the formulas in Γ implies the OR of all the formulas in Δ. The size and depth of a cedent are respectively the sum of sizes and the maximum of depths of all the formulas in it. The size of a sequent is the sum of sizes of both cedents, and the depth is the maximum of the depths of both cedents.
Definition of a Proof. A proof in this system is defined as a sequence of sequents $\mathcal{S}_1 \cdots \mathcal{S}_m$ such that each $\mathcal{S}_i$ is either an initial sequent, or is derived from sequents $\mathcal{S}_j$ for $j < i$ through one of the rules listed below. The size and depth of a proof are respectively the sum of sizes and the maximum of depths of all sequents in it.

The initial sequents and the derivation rules are listed below.

Translating lines. We will now define translations of lines in the above proof system. For a formula φ, we denote its translation in depth-d-PC by $tr(\varphi)$. Let $x_1 \cdots x_n$ be the variables of the original proof. Below we list the translations for a formula built with each connective. The interpretation is that for any formula φ, $tr(\varphi) = 0$ if and only if φ is true.

$tr(x_i) = 1 - x_i$
$tr(\vee(\varphi_1 \cdots \varphi_k)) = \prod_i (tr(\varphi_i))$
$tr(\wedge(\varphi_1 \cdots \varphi_k)) = 1 - \prod_i tr(\neg\varphi_i)$
$tr(\oplus_i^p(\varphi_1 \cdots \varphi_k)) = (\sum_{j=1}^{k} tr(\varphi_j) - i)^{p-1}$ for $0 \leq i < p$
$tr(Th_t(\varphi_1 \cdots \varphi_k)) = (y - \alpha^t) \cdots (y - \alpha^k)$
where $y = \prod_i ((\alpha - 1)tr(\neg\varphi_i) + 1)$
$tr(\neg\varphi) = 1 - tr(\varphi)$ if φ does not contain a Th_t connective
$tr(\neg Th_t(\varphi_1 \cdots \varphi_k)) = (y - 1) \cdots (y - \alpha^{t-1})$
where $y = \prod_i ((\alpha - 1)tr(\neg\varphi_i) + 1)$, for $t \geq 1$
The translation $tr(\mathcal{S})$ of a sequent $\mathcal{S}$ of the form $\varphi_1 \cdots \varphi_k \rightarrow \varphi'_1 \cdots \varphi'_{k'}$ is given by the equation

$$\prod_{i=1}^{k} tr(\neg\varphi_i) \prod_{j=1}^{k'} tr(\varphi'_j) = 0$$

Note that the translations of all the connectives except the threshold connective take only boolean values over $\mathbb{F}_{p^m}$.

Simulating proofs. We now describe the connection between $\mathsf{AC}^0[p]$-Frege and the proof system of Maciel and Pitassi. By the following theorem of Allender [2], any $\mathsf{AC}^0[p]$ circuit can converted to a depth three circuit of a special form.

Theorem 6 ([2]). *Any $\mathsf{AC}^0[p]$ circuit can be converted to a quasipolynomial sized depth three circuit with an unweighted threshold gate at the top, MOD_p gates of quasipolynomial fan-in in the middle and $\wedge$ gates of polylogarithmic fan-in at the bottom*

Depth three circuits with an unweighted threshold, $\wedge$ or $\vee$ gate at the top, MOD_p gates in the middle and $\wedge$ gates of polylogarithmic fan-in in the size of the circuit at the bottom are referred to as *flat circuits* by [21]. For an $\mathsf{AC}^0[p]$ circuit φ, its *flattening* $fl(\varphi)$ is defined as the flat circuit given by the above theorem. Proofs in $\mathsf{AC}^0[p]$-Frege can be thought of as a list of sequents such that every formula that appears in each of them is an $\mathsf{AC}^0[p]$ circuit. For a sequent $\varphi_1 \cdots \varphi_k \rightarrow \varphi'_1 \cdots \varphi'_{k'}$ that appears in a $\mathsf{AC}^0[p]$-Frege proof, we can define a flattening of the sequent $fl(\varphi_1) \cdots fl(\varphi_k) \rightarrow fl(\varphi'_1) \cdots fl(\varphi'_{k'})$ in the proof system of Maciel and Pitassi. A *flat proof* of such a sequent is such that every formula that appears in the proof is a flat circuit. The simulation theorem of [21] states the following

Theorem 7 ([21]). *Let $\mathcal{S}$ be a sequent which has a depth d proof in $\mathsf{AC}^0[p]$-Frege. Then its flattening $fl(\mathcal{S})$ has a flat proof of size $2^{(\log n)^{O(d)}}$ in the proof system of Maciel and Pitassi.*

We will show that flat proofs can be simulated in depth-d-PC by showing the following

Theorem 8. *Let $\mathcal{S}$ be a sequent which has a flat proof of size s in the proof system of Maciel and Pitassi. Then there is a proof of the equation $tr(\mathcal{S})$ in depth-d-PC from the equations $x_i(x_i - 1) = 0$ with $poly(s)$ lines.*

To prove the above theorem, it is sufficient to show that for each rule that derives a sequent $\mathcal{S}_3$ from sequents $\mathcal{S}_1$ and $\mathcal{S}_2$, there is a derivation of the equation $tr(\mathcal{S}_3)$ from the equations $tr(\mathcal{S}_1)$, $tr(\mathcal{S}_2)$ and $x_i(x_i - 1) = 0$ in depth-d-PC. The details of how each such rule can be simulated are left to the full version [18].

4.2.2 Case of $q \neq p$. We now extend the simulation of the previous section to show that $\mathsf{AC}^0[q]$-Frege can be simulated in depth-d-PC over F_{p^m}, for distinct primes p and q, hence proving Theorem 2. Using the theorem of Maciel and Pitassi (Theorem 7 above) for $\mathsf{AC}^0[q]$-Frege, we obtain a flat proof with $\oplus_i^q$ connectives. To simulate it, we can reuse the lemmas of the previous section, except for the $\oplus_i^q$ connectives. To define their translation, choose m such that $q \mid p^m - 1$ and let $r = (p^m - 1)/q$. The translation is now defined as

$$tr(\oplus_i^q(\varphi_1 \cdots \varphi_k)) = \left((y - \alpha^{ir})\right)^{p^m - 1}$$

where $y = \prod_i ((\alpha^r - 1)tr(\neg\varphi_i) + 1)$ and $tr(\neg\oplus_i^q(\varphi_1 \cdots \varphi_k)) = 1 - tr(\oplus_i^q(\varphi_1 \cdots \varphi_k))$

Simulating the rules is similar to the previous section. See the full version [18] for more details.

4.3 Simulating TC^0-Frege

In this section, we show that a TC^0-Frege proof of depth d_0 can be transformed into a depth-d-PC proof over $\mathbb{F}_{p^m}$, where $d = O(d_0)$, proving Theorem 3. In the previous section we translated $Th_t(\varphi_1 \cdots \varphi_k)$ as

$$tr(Th_t(\varphi_1 \cdots \varphi_k)) = (y - \alpha^t) \cdots (y - \alpha^k)$$

$$tr(\neg Th_t(\varphi_1 \cdots \varphi_k)) = (y - 1) \cdots (y - \alpha^{t-1})$$

where $y = \prod_i ((\alpha - 1)tr(\neg\varphi_i) + 1)$. Clearly this translation requires $tr(\varphi_i)$ to be boolean and can itself take non-boolean values. Since there is only one top threshold gate in a flat circuit, the formulae φ_i were threshold free and thus $tr(\varphi_i)$ only took on boolean values. But in a TC^0-Frege proof, the formulae φ_i can themselves contain threshold gates and thus $tr(\varphi_i)$ may be non-boolean. To fix this problem, we redefine the translation of a threshold gate to be the following, essentially forcing it to be boolean.

$$tr(Th_t(\varphi_1 \cdots \varphi_k)) = \left((y - \alpha^t) \cdots (y - \alpha^k)\right)^{p^m - 1}$$

where $y = \prod_i ((\alpha - 1)tr(\neg\varphi_i) + 1)$ and $tr(\neg Th_t(\varphi_1 \cdots \varphi_k)) = 1 - tr(Th_t(\varphi_1 \cdots \varphi_k))$.

It is easy to derive the fact that the above translation only takes boolean values. Now, note that any rule other than the Th_t is unaffected by this new translation since it only assumes that its arguments are boolean and hence we can use the lemmas of the previous section directly. However, simulation of the Th_t rule relies on the old translation. To bridge the gap, we only need to show that the old and new translations of Th_t and $\neg Th_t$ are interchangeable within the proof system. The following lemmas are proved in the full version [18].

Lemma 2. *Given the equation*

$$\left((y - \alpha^t) \cdots (y - \alpha^k)\right)^{p^m - 1} = 0$$

we can derive

$$(y - \alpha^t) \cdots (y - \alpha^k) = 0$$

and vice versa.

Lemma 3. *Given the equation*

$$1 - \left((y - \alpha^t) \cdots (y - \alpha^k)\right)^{p^m - 1} = 0$$

we can derive

$$(y - 1) \cdots (y - \alpha^{t-1}) = 0$$

and vice versa.

4.3.1 Existence of Feasible Interpolation.

Bonet, Pitassi and Raz [4] have shown that TC^0-Frege does not have feasible interpolation unless Blum integers can be factored by polynomial sized circuits. By the above simulation, we can state the following

Theorem 9. *Depth-d-PC does not have feasible interpolation unless Blum integers can be factored by polynomial sized circuits*

4.4 Dealing with large coefficients – Simulating CP and Dynamic SOS

In this section, we work over a field $\mathbb{F}_{p^m}$ for an arbitrary prime p, where p^m is greater than square of the number of monomials we wish to represent in any CP/SOS proof line (See Definition 17).

It is well-known that arbitrary threshold gates can be simulated by simple majority gates of higher depth. In particular, a tight simulation was proven by Goldmann, Hastad and Razborov [12] who show that depth $d + 1$ TC^0 circuits are equivalent to depth d threshold circuits with arbitrary weights. However, the analogous result has not been proven in the propositional proof setting. In order to simulate arbitrary weighted thresholds in our low depth extension of PC, we will we use a different simulation of high weight thresholds by low weight ones.

The basic idea will be to use simple, shallow formulas that compute the iterated addition of n binary numbers, each with $\xi = \mathsf{poly}(n)$ bits [22]. Let $\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_n}$ be the set of n binary numbers, each of length $\xi = \mathsf{poly}(n)$, where $\mathbf{a_i} = a_{i,\xi}, \cdots, a_{i,1}$. We will break up the ξ coordinates into $\xi/\log\xi$ blocks, each of size $\log\xi$; let $L_j(\mathbf{a_i})$ denote the j^{th} block of $\mathbf{a_i}$. The high level idea is to compute the sum by first computing the sum *within* each block, and then to combine using carry-save-addition.

In more detail, let $\mathbf{a_i^o}$ denote the "odd" blocks of $\mathbf{a_i}$ – so $\mathbf{a_i^o}$ consists of $\xi/\log\xi$ blocks, where for j odd, the j^{th} block is $L_j(\mathbf{a_i})$, and for j even, the j^{th} block is all zeroes (and similarly, $\mathbf{a_i^e}$ denotes the even blocks of $\mathbf{a_i}$). Let S^o be equal to $\sum_{i\in[n]} \mathbf{a_i^o}$, and similarly let S^e be equal to $\sum_{i\in[n]} \mathbf{a_i^e}$. We will give a SLP for computing the bits of S^o and S^e and then our desired sum, $S^o + S^e$, is obtained using the usual carry-save addition which can be computed by a depth-2 SLP. The main point is that we have padded $\mathbf{a_i^o}$ and $\mathbf{a_i^e}$ with zeroes in every other block; this enables us to compute S^o (and similarly S^e) *blockwise* (on the odd blocks for S^o and on the even blocks for S^e), because no carries will spill over to the next nonzero block. Then since the blocks are very small ($\log\xi$ bits), the sum within each block can be carried out by brute-force.

Our construction below generalizes this to the case where the $\mathbf{a_i}$'s are not large coefficients, but instead they are the product of a monomial and a large coefficient. After formally describing this low-depth representation, it remains to show how to efficiently reason about these low-depth representations in order to carry out the rule-by-rule simulation of general Cutting Planes and SOS. We outline the main steps below, with technical details left to the full version [18].

4.4.1 Bit vector representations of CP/SOS proof lines.

Definition 12 (Derivations in depth-d-PC). *To indicate that a new extension variable y_i is being introduced and set to a value a_i, we write*

$$y_i := a_i$$

To indicate that a line $P = 0$ in depth-d-PC can be derived from $P_1 = 0, P_2 = 0, \cdots, P_k = 0$, we write

$$P_1, P_2, \cdots, P_k \vdash P$$

To indicate that a line $P = 0$ can be derived just from the axioms of the form $x_i^2 = x_i$ for all boolean variables x_i, we write

$$\vdash P$$

Below we formally define the representation of binary numbers as bit vectors.

Definition 13 (Bit vectors). *We represent an integer using its bit representation by introducing a variable for each of its bits. Let a be an integer with bits $a_\xi \cdots a_1$. A bit vector $\mathbf{a} = [a_\xi \cdots a_1]$ representing the integer a in our system is a set of auxiliary variables $y_\xi \cdots y_1$ such that $y_i := a_i$. Define $\mathbf{a}(i) = y_i = a_i$. Integers which are represented as vectors are written in boldface.*

Let ξ_0 be an upper limit on the number of monomials in any polynomial we wish to represent and let ξ_1 be an upper limit on any coefficient we wish to represent. Set $\xi = 10\lceil log(\xi_0) + log(\xi_1)\rceil$. The bit vectors in this simulation will all be of dimension ξ, i.e. all integers we represent will be of at most ξ bits. Any vector of dimension $> \xi$ generated in any operation is automatically truncated to dimension ξ by dropping the higher order bits.

The bit representation chosen is two's complement. That is, a positive integer is represented in binary in the usual way. Let b be a positive integer represented by $\mathbf{b}$. Let $\mathbf{b}_1$ be the vector obtained by flipping all the bits in $\mathbf{b}$. Then we define the vector $-\mathbf{b}$ as $\mathbf{b}_1 \oplus \mathbf{1}$, where $\oplus$ operation on vectors, defined below, simulates the usual bitwise addition operation and $\mathbf{1}$ is the vector representation of the integer 1. $\mathbf{0}$, the all zeros vector, represents the integer 0. For any vector $\mathbf{a}$, $\mathbf{a}(\xi)$ is the sign bit of $\mathbf{a}$. $\mathbf{a}$ is said to be negative if the sign bit is one.

In order to make correct computation using the above Two's complement representation of binary numbers, we need to ensure that the bit length of all numbers represented is bounded. We therefore define the *length* of a vector in our simulation, and later show that such vectors are of bounded length.

Definition 14 (Length of a vector). *The length of a non-negative vector $\mathbf{a}$ is the highest index i such that $\mathbf{a}(i) \neq 0$ and zero if such an i does not exist. The length of a negative vector $\mathbf{b}$ is the highest index i such that $\mathbf{b}(i) \neq 1$. Equivalently, the length of a vector $\mathbf{a}$ is the highest index i such that $\mathbf{a}(i) \neq \mathbf{a}(\xi)$.*

We now define the usual addition operation for binary numbers, over their vector representations. Since we work in a low depth setting, we need to use *Carry-Save* addition to represent the sum and carry bits.

4.4.2 Operations on bit vectors.

Definition 15 (The Bitwise Addition operation $\oplus$). *We define below the operator on vectors corresponding to the usual carry-save addition. For two bits y and z, let $y \oplus z$ represent the XOR of the bits. Given two bit vectors $\mathbf{y} = [y_\xi \cdots y_1]$ and $\mathbf{z} = [z_\xi \cdots z_1]$, the bitwise addition operation $\mathbf{y} \oplus \mathbf{z}$ produces a vector $[w_{\xi+1} \cdots w_1]$ such that*

$$w_i := y_i \oplus z_i \oplus c_i$$

for $i \leq \xi$ and $w_{\xi+1} := c_\xi$ where

$$c_i := \vee_{j<i}(y_j \wedge z_j \wedge_{j<k<i} (y_k \oplus z_k))$$

for $1 < i \leq \xi$ and $c_1 := 0$.

c_i are referred to as the carry bits in $\mathbf{y} \oplus \mathbf{z}$

Monomial terms a_1X_1 in our system are represented by a "scalar multiplication" of X_1 with the vector $\mathbf{a}_1$, which we define below.

Definition 16 (Scalar multiplication). *For a bit z and a vector $\mathbf{y}$, let $z\mathbf{y} = \mathbf{y}z$ represent the vector obtained by multiplying every bit of $\mathbf{y}$ by z.*

In order to represent a line $a_1X_1 + \cdots + a_nX_n - a_0 \geq 0$ in Cutting Planes, we define an operation $\mathcal{S}$ over the vectors $\mathbf{a}_1X_1, \cdots, \mathbf{a}_nX_n$ such that the resultant vector is a representation of $a_1X_1 + \cdots + a_nX_n - a_0$ and has a low depth in $X_1, \cdots, X_n$. This uses the idea of representing high weight thresholds using low depth majority gates described earlier.

Definition 17 (The Set Addition operation $\mathcal{S}(.)$). *We will now define the representation of the bitwise addition of vectors $\mathbf{a}_1X_1, \cdots, \mathbf{a}_tX_t$, where $\mathbf{a}_1, \cdots, \mathbf{a}_t$ are integer constants and $X_1, \cdots, X_t$ are monomials.*

Let $\xi_2 = \lceil \xi/\log(\xi_0)\rceil$. For a constant $\mathbf{a}$, partition the bits of $\mathbf{a}$ into ξ_2 blocks of length at most $\log(\xi_0)$. Let $L_j(\mathbf{a})$, $j \in [\xi_2]$ denote the j^{th} block of bits, so that the bits of $\mathbf{a}$ can be obtained by a concatenation of the bits $L_{\xi_2}(\mathbf{a})...L_1(\mathbf{a})$. Since $L_j(\mathbf{a})$ is only $\log(\xi_0)$ bits long, its magnitude is at most ξ_0. Let $[L_j(\mathbf{a})]$ refer to the integer represented by the vector $L_j(\mathbf{a})$. Define $\mathbf{a}^o$ to be the vector obtained by replacing all even numbered blocks of $\mathbf{a}$ with zeroes. $\mathbf{a}^e$ is analogously defined by zeroing out the odd numbered blocks. For monomials $X_1 \cdots X_t$ and $t < \xi_0$, we would like to define bit vectors $\mathcal{S}^o(\mathbf{a}_1X_1, \cdots, \mathbf{a}_tX_t)$ and $\mathcal{S}^e(\mathbf{a}_1X_1, \cdots, \mathbf{a}_nX_t)$ to be the bit representations of the polynomials $\sum_{i=1}^t a_i^o X_i$ and $\sum_{i=1}^t a_i^e X_i$. We accomplish this using constant depth SLPs as follows.

We define a constant depth SLP to compute the k^{th} bit of the j^{th} block of $\mathcal{S}^o$, represented by $L_{jk}(\mathcal{S}^o)$. The important observation is that we can compute $\mathcal{S}^o$ two blocks at a time

since for odd j, $\sum_i [L_j(\mathbf{a}_i^o)]X_i$ is at most ξ_0^2 and thus can be represented by $2\log(\xi_0)$ bits or exactly two blocks. Let C_ℓ be the set of integers in $[\xi_0^2]$ such that the ℓ^{th} bit of their binary representation is one. Then for odd j, $L_{jk}(\mathcal{S}^o)$ is one if and only if

$$\prod_{\beta \in C_k} \left(\sum_i [L_j(\mathbf{a}_i^o)]X_i - \beta \right) = 0$$

and for even j, $L_{jk}(\mathcal{S}^o)$ is one if and only if

$$\prod_{\beta \in C_{\log(\xi_0)+k}} \left(\sum_i [L_{j-1}(\mathbf{a}_i^o)]X_i - \beta \right) = 0$$

Therefore, the bit $L_{jk}(\mathcal{S}^o)$ can be represented as a constant depth SLP of size $O(\xi_0)$ by representing the left hand side of the above equations as a SLP over a finite field extension larger than ξ_0^2, similar to the simulation of CP in the earlier sections, and then raising the result of that SLP to the order of the multiplicative group that we are working in. The bits of $\mathcal{S}^e$ are represented analogously.*

The operation $\mathcal{S}$ over vectors $\mathbf{a}_1X_1, \cdots, \mathbf{a}_tX_t$ is now defined as $\mathcal{S}^o(\mathbf{a}_1X_1, \cdots, \mathbf{a}_tX_t) \oplus \mathcal{S}^e(\mathbf{a}_1X_1, \cdots, \mathbf{a}_nX_t)$.

4.4.3 Representing a line from CP/SOS in depth-d-PC.

We now define the translation of a line $a_1X_1 + \cdots + a_nX_k - a_0 \geq 0$ in Cutting Planes/SOS, where $X_1 \ldots X_k$ are monomials.

Definition 18 (Representing an inequality)**.** *Let $P = a_1X_1 + \cdots + a_kX_k$ be a polynomial where the X_i are monomials. Then the line $P \geq 0$ is represented as*

$$\mathcal{S}(\mathbf{a}_1X_1, \cdots, \mathbf{a}_kX_k)(\xi) = 0$$

and $P = 0$ is represented as

$$\mathcal{S}(\mathbf{a}_1X_1, \cdots, \mathbf{a}_kX_k) = \mathbf{0}$$

Let $\mathcal{R}(P)$ denote the vector $\mathcal{S}(\mathbf{a}_1X_1, \cdots, \mathbf{a}_kX_k)$.

4.4.4 Simulating Cutting Planes.

Addition. Before we prove the simulation for addition, we need the following key properties of the vector representation. They are proved in the full version [18].

The lemma below states that our system can prove the associativity of the operation $\oplus$ over vectors.

Lemma 4. *For any three bit vectors $\mathbf{y}$, $\mathbf{z}$ and $\mathbf{w}$*

$$\vdash (\mathbf{y} \oplus \mathbf{z}) \oplus \mathbf{w} - \mathbf{y} \oplus (\mathbf{z} \oplus \mathbf{w})$$

We then need to be able to interchangeably use the operations $\mathcal{S}$ and $\oplus$ for vector addition

Lemma 5. $\vdash \mathcal{S}(\mathbf{y}_1, \cdots, \mathbf{y}_i) - \mathcal{S}(\mathbf{y}_1, \cdots, \mathbf{y}_{i-1}) \oplus \mathbf{y}_i$

We then extend this to show that the vector representation of the sum of two lines is the $\oplus$ of the vector representations of each line.

Lemma 6. *Let P and Q be two polynomials. Then $\mathcal{R}(P+Q) = \mathcal{R}(P) \oplus \mathcal{R}(Q)$.*

Finally, we need to show that the as long as P and Q have coefficients not exceeding bit length ξ, we can derive from $\mathcal{R}(P)(\xi) = 0$ and $\mathcal{R}(Q)(\xi) = 0$ the lines $\mathcal{R}(P+Q)(\xi) = 0$. It is an easy observation that if the bit lengths of the coefficients in P and Q are bounded, then the vectors $\mathcal{R}(P)$ and $\mathcal{R}(Q)$ are of bounded length. Thus it suffices to show the following.

Lemma 7. *For any two vectors $\mathbf{a}$ and $\mathbf{b}$ of length at most $\ell < \xi - 1$*

$$\mathbf{a}(\xi), \mathbf{b}(\xi) \vdash (\mathbf{a} \oplus \mathbf{b})(\xi)$$

This concludes simulation of the addition rule.

Multiplication by a constant. In order to simulate multiplication by a power of two, we left-shift bits of the corresponding bit vector by the required amount, and add zero bits at the end. Multiplication by any constant can then be simulated by the above in combination with the Addition rule.

Division by a constant. To simulate the division rule in Cutting Planes we use the following lemma.

Lemma 8. *Let $P = a_1x_1 + \cdots + a_nx_n - a_0$ where a_i are non-negative, $a_1 \cdots a_n$ are even and a_0 is odd. Then we can derive*

$$\mathcal{R}(P)(\xi) \vdash \mathcal{R}(P-1)(\xi)$$

Proof. It is easy to derive

$$\mathbf{a}_0(1) - 1 \vdash (-\mathbf{a}_0)(1) - 1$$

Since we have $\vdash \mathcal{R}(P) - \left(\mathcal{S}(\mathbf{a}_1x_1, \cdots, \mathbf{a}_nx_n) \oplus (-\mathbf{a}_0)\right)$ by Lemma 5, and $a_1 \cdots a_n$ are even, we derive

$$\vdash \mathcal{R}(P)(1) - 1$$

Since -1 is represented by the all ones vector, for every carry bit c_i in the sum $\mathcal{R}(P) \oplus (-\mathbf{1})$ it is easy to derive from the definition of c_i

$$\vdash c_i - 1$$

Now using the definition $(\mathcal{R}(P) \oplus (-\mathbf{1}))(\xi) = \mathcal{R}(P)(\xi) \oplus 1 \oplus c_\xi$ and Lemma 5 we derive

$$\mathcal{R}(P)(\xi) \vdash \mathcal{R}(P-1)(\xi)$$

□

We can now simulate the division rule by using the above lemma and then dropping the last bit of the vector $\mathcal{R}(P-1)$ (which would be zero).

4.4.5 Simulating Dynamic SOS.

Rules 1, 2 and 3 of Definition 7 follow from the above simulation of Cutting Planes.

Multiplication of two lines. To simulate the multiplication rule of SOS, we need to define an operation which, given the vectors $\mathbf{a}_1$ and $\mathbf{b}_1$, produces a vector that is equivalent to the representation of $a_1 b_1$. We define it as a shifted sum based on the grade school algorithm for binary multiplication.

Definition 19 (Shifted sum)**.** *For a vector* $\mathbf{y}$*, let* $2^k\mathbf{y}$ *denote the vector obtained by shifting the bits of* $\mathbf{y}$ *to the left by* k *positions, and padding the least significant* k *positions with zeros. Given two vectors* $\mathbf{y}$ *and* $\mathbf{z} = [z_{\xi-1} \cdots z_0]$ *, the shifted sum of* $\mathbf{y}$ *and* $\mathbf{z}$ *is defined as the vector*

$$\mathcal{SS}(\mathbf{y}, \mathbf{z}) = \mathcal{S}(z_0\mathbf{y}, \cdots, z_{\xi-1}2^{\xi-1}\mathbf{y})$$

We then show that our system can prove that the vector obtained by using this operation is indeed what we want.

Lemma 9. *Let* P *and* Q *be two polynomials, represented by bit vectors* $\mathbf{y}_0$ *and* $\mathbf{z} = [z_{\xi-1} \cdots z_0]$*, with at most* ξ_0 *monomials and coefficients bounded by* ξ_1 *in absolute value. Then,*

$$\vdash \mathcal{R}(PQ) - \mathcal{SS}(\mathbf{y}_0, \mathbf{z})$$

We now extend Lemma 7 to show that we can derive $PQ \geq 0$ from $P \geq 0$ and $Q \geq 0$, i.e. $\mathcal{R}(PQ)(\xi) = 0$ from $\mathcal{R}(P)(\xi) = 0$ and $\mathcal{R}(Q)(\xi) = 0$.

Lemma 10. *Let* $\mathbf{y}$ *and* $\mathbf{z}$ *be two non-negative vectors of length* ℓ *such that* $3\ell < \xi - 1$*. Then*

$$\mathbf{y}(\xi), \mathbf{z}(\xi) \vdash \mathcal{SS}(\mathbf{y}, \mathbf{z})(\xi)$$

This completes the simulation of the rule which takes the product of two lines in SOS.

Squaring rule. To simulate the rule in SOS which introduces a line $P^2 \geq 0$ for any polynomial P, we need the following lemmas.

The lemma below states that if the sign bit of $\mathbf{y}$ is one, then the sign bit of $-\mathbf{y}$ is zero.

Lemma 11. *For any vector* $\mathbf{y}$ *of length* $\ell < \xi - 1$*,*

$$\mathbf{y}(\xi) - 1 \vdash (-\mathbf{y})(\xi)$$

The following lemma shows that for a vector representing a polynomial P, the negation of it represents the polynomial $-P$.

Lemma 12. *Let* P *be a polynomial represented by a vector* $\mathbf{y}$*. Then* $\vdash \mathcal{R}(-P) - (-\mathbf{y})$*.*

The rule which derives $P^2 \geq 0$ can now be easily simulated by branching on the sign bit of the vector $\mathcal{R}(P)$. Assuming it to be zero, we can use Lemma 10 to derive $\mathcal{R}(P^2)(\xi) = 0$. In the other case, we can use Lemma 11 and Lemma 12 to derive that the sign bit of $\mathcal{R}(-P)$ is zero. We can now use Lemma 10 again to derive $\mathcal{R}(P^2)(\xi) = 0$.

4.4.6 Concluding the simulation. By simulating any refutation in Cutting Planes/SOS rule by rule using the above lemmas, we end up with the representation of the line $-1 \geq 0$ i.e.

$$\mathcal{R}(-1)(\xi) = 0$$

Since -1 is represented by the all ones vector, this gives a contradiction.

Open Problems

The obvious open problem is to prove a lower bound for $\mathsf{AC}^0[\mathsf{p}]$-Frege systems, whether using algebraic proofs or not.

As stepping stones towards this goal, we think it would be interesting to:

1. Find any technique for proving lower bounds on the sizes of Polynomial Calculus proofs that doesn't go through degrees. More precisely, prove size lower bounds for PC proofs where we view variables as taking values $1, -1$, and replace the axioms $x^2 - x$ with $x^2 - 1$.
2. Prove lower bounds for the system Trinomial-ΠΣ-PC.
3. Our simulations require a sufficiently large extension field. Can we either p-simulate Polynomial Calculus over a large extension field with Polynomial Calculus over the base field, or prove that no simulation exists?

Acknowledgements

The authors would like to thank Paul Beame, Lijie Chen, Srikanth Srinivasan and Iddo Tzameret for helpful discussions. Part of this research took place at the Simons Institute at UC Berkeley, and we are thankful for their support.

References

[1] Yaroslav Alekseev, Dima Grigoriev, Edward A Hirsch, and Iddo Tzameret. 2019. Semi-Algebraic Proofs, IPS Lower Bounds and the τ-Conjecture: Can a Natural Number be Negative? *arXiv preprint arXiv:1911.06738* (2019).

[2] Eric Allender. 1989. A note on the power of threshold circuits. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*. IEEE, 580–584.

[3] Paul Beame, Russell Impagliazzo, Jan Krajícek, Toniann Pitassi, and Pavel Pudlák. 1994. Lower bounds on Hilbert's Nullstellensatz and propositional proofs. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE, 794–806.

[4] Maria Luisa Bonet, Toniann Pitassi, and Ran Raz. 2000. On interpolation and automatization for Frege systems. *SIAM J. Comput.* 29, 6 (2000), 1939–1967.

[5] Sam Buss, Dima Grigoriev, Russell Impagliazzo, and Toniann Pitassi. 2001. Linear gaps between degrees for the polynomial calculus modulo distinct primes. *J. Comput. System Sci.* 62, 2 (2001), 267–289.

[6] Samuel Buss, Leszek Kołodziejczyk, and Konrad Zdanowski. 2015. Collapsing modular counting in bounded arithmetic and constant depth propositional proofs. *Trans. Amer. Math. Soc.* 367, 11 (2015), 7517–7563.

[7] Samuel R Buss and Peter Clote. 1996. Cutting planes, connectivity, and threshold logic. *Archive for Mathematical Logic* 35, 1 (1996), 33–62.

[8] Samuel R. Buss, Russell Impagliazzo, Jan Krajícek, Pavel Pudlák, Alexander A. Razborov, and Jirí Sgall. 1997. Proof Complexity in Algebraic Systems and Bounded Depth Frege Systems with Modular Counting. *Computational Complexity* 6, 3 (1997), 256–298.

[9] Matthew Clegg, Jeffery Edmonds, and Russell Impagliazzo. 1996. Using the Groebner basis algorithm to find proofs of unsatisfiability. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM, 174–183.

[10] Peter Clote and Evangelos Kranakis. 2013. *Boolean functions and computation models*. Springer Science & Business Media.

[11] William Cook, Collette R Coullard, and Gy Turán. 1987. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics* 18, 1 (1987), 25–38.

[12] Mikael Goldmann, Johan Håstad, and Alexander A. Razborov. 1992. Majority Gates VS. General Weighted Threshold Gates. *Computational Complexity* 2 (1992), 277–300.

[13] Dima Grigoriev and Edward A Hirsch. 2003. Algebraic proof systems over formulas. *Theoretical Computer Science* 303, 1 (2003), 83–102.

[14] Dima Grigoriev and Nicolai Vorobjov. 2001. Complexity of Null-and Positivstellensatz proofs. *Annals of Pure and Applied Logic* 113, 1-3 (2001), 153–160.

[15] Joshua A Grochow and Toniann Pitassi. 2014. Circuit complexity, proof complexity, and polynomial identity testing. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*. IEEE, 110–119.

[16] Joshua A. Grochow and Toniann Pitassi. 2018. Circuit Complexity, Proof Complexity, and Polynomial Identity Testing: The Ideal Proof System. *J. ACM* 65, 6 (2018), 37:1–37:59.

[17] Pavel Hrubes and Iddo Tzameret. 2009. The Proof Complexity of Polynomial Identities. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009, Paris, France, 15-18 July 2009*. 41–51.

[18] Russell Impagliazzo, Sasank Mouli, and Toniann Pitassi. 2019. The Surprising Power of Constant Depth Algebraic Proofs.. In *Electronic Colloquium on Computational Complexity (ECCC)*, Vol. 26. 1–4.

[19] Russell Impagliazzo, Pavel Pudlák, and Jiri Sgall. 1999. Lower bounds for the polynomial calculus and the Gröbner basis algorithm. *Computational Complexity* 8, 2 (1999), 127–144.

[20] Jan Krajíček. 1998. Discretely ordered modules as a first-order extension of the cutting planes proof system. *The Journal of Symbolic Logic* 63, 4 (1998), 1582–1596.

[21] Alexis Maciel and Toniann Pitassi. 1998. Towards lower bounds for bounded-depth Frege proofs with modular connectives. *Proof complexity and feasible arithmetics* 39 (1998), 195–227.

[22] Alexis Maciel and Denis Thérien. 1998. Threshold Circuits of Small Majority-Depth. *Inf. Comput.* 146, 1 (1998), 55–83.

[23] Toniann Pitassi. 1996. Algebraic Propositional Proof Systems. In *Descriptive Complexity and Finite Models, Proceedings of a DIMACS Workshop 1996, Princeton, New Jersey, USA, January 14-17, 1996*. 215–244.

[24] Toniann Pitassi. 1998. Unsolvable systems of equations and proof complexity. In *Proceedings of the International Congress of Mathemeticians, Volume III, Berlin*. 451–460.

[25] Toniann Pitassi and Rahul Santhanam. 2010. Effectively Polynomial Simulations.. In *ICS*. 370–382.

[26] Ran Raz and Iddo Tzameret. 2008. Resolution over linear equations and multilinear proofs. *Ann. Pure Appl. Logic* 155, 3 (2008), 194–224.

[27] Ran Raz and Iddo Tzameret. 2008. The strength of multilinear proofs. *computational complexity* 17, 3 (2008), 407–457.

[28] Alexander A Razborov. 1987. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes* 41, 4 (1987), 333–338.

[29] Alexander A Razborov. 1998. Lower bounds for the polynomial calculus. *computational complexity* 7, 4 (1998), 291–324.

[30] Roman Smolensky. 1987. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM, 77–82.