Electronic Colloquium on Computational Complexity, Revision 2 of Report No. 24 (2019)

The Surprising Power of Constant Depth Algebraic Proofs

Russell Impagliazzo
russell@cs.ucsd.edu

Sasank Mouli
sasankm@ucsd.edu

Toniann Pitassi
toni@cs.utoronto.edu

Abstract

A major open problem in proof complexity is to prove superpolynomial lower bounds for $\mathtt{AC}^0[\mathtt{p}]$-Frege proofs. This system is the analog of $\mathtt{AC}^0[\mathtt{p}]$, the class of bounded depth circuits with prime modular counting gates. Despite strong lower bounds for this class dating back thirty years ([27, 29]), there are no significant lower bounds for $\mathtt{AC}^0[\mathtt{p}]$-Frege. Significant and extensive *degree* lower bounds have been obtained for a variety of subsystems of $\mathtt{AC}^0[\mathtt{p}]$-Frege, including Nullstellensatz ([3]), Polynomial Calculus ([9]), and SOS ([14]). However to date there has been no progress on $\mathtt{AC}^0[\mathtt{p}]$-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 simulate all of the well-studied semialgebraic proof systems: Cutting Planes, Sherali-Adams, Sum-of-Squares (SOS), and Positivestellensatz Calculus (Dynamic SOS). Additionally, they can also quasi-polynomially simulate $\mathtt{AC}^0[\mathtt{q}]$-Frege for *any* prime q independent of the characteristic of the underlying field. They can also simulate $\mathtt{TC}^0$-Frege if the depth is allowed to grow proportionally. Thus, proving strong lower bounds for $\mathtt{AC}^0[\mathtt{p}]$-Frege would seem to require proving lower bounds for systems as strong as $\mathtt{TC}^0$-Frege.

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 $\mathtt{AC}^0[\mathtt{p}]$, the class of bounded depth circuits with prime modular counting gates. Despite strong lower bounds for this class dating back thirty years ([27, 29]), there are no significant lower bounds for $\mathtt{AC}^0[\mathtt{p}]$-Frege. Since the only lower bounds for circuits with modular operations are via

ISSN 1433-8092

representations of functions by polynomials ([27, 29]), it seems natural to use algebraic proof systems (e.g, Nullstellensatz ([3]), Polynomial Calculus (PC) ([9]), Positivestellensatz 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 $\mathtt{AC^0[p]}$-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 $\mathtt{AC^0[p]}$ circuits.

Since lower bounds for Polynomial Calculus itself do not imply lower bounds for $\mathtt{AC^0[p]}$-Frege systems, various researchers have suggested ways to strengthen PC to create algebraic systems which do p-simulate $\mathtt{AC^0[p]}$-Frege ([22, 13, 8]). 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 [28], [18] 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 $\mathtt{AC^0[p]}$-Frege proofs by using arithmetic circuits of depth $d' = \Theta(d)$. Raz and Tzameret [26] 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 [25] that Polynomial Calculus with added linear transformations simulates the system $R(CP^*)$ of Krajicek [19], 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 $\mathtt{AC^0[p]}$ proofs also allows it to simulate much stronger proof systems. So to prove a lower bound on $\mathtt{AC^0[p]}$ proofs via such systems would seem to require proving lower bounds for systems as strong as $\mathtt{TC^0}$-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.)

All our simulation results below use the notion of *effective simulation* from [24] (see Definition 4). For the rest of the paper, "simulate" refers to an effective simulation.

We remove the restriction of polynomially bounded coefficients from the result of [25] 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 Positivestellensatz Calculus [14]).

Theorem 1. *Depth-43-PC can effectively* $\mathtt{p}$*-simulate Cutting Planes and Positivestellensatz 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. Theorem 1 is proved in section 5.4.

We improve the results of Grigoriev and Hirsch in the constant depth case in two ways. We show that $\mathtt{AC^0[p]}$-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* $m = O(\mathtt{poly}(\log(\mathtt{n})))$*, depth-9-PC over* $\mathbb{F}_{p^m}$ *can effectively quasipolynomially simulate* $\mathtt{AC^0[q]}$*-Frege over* n *variables for any prime* q*.*

Buss *et al.* [6] showed that an $\mathtt{AC^0[p]}$-Frege proof of depth d can be collapsed to a depth 3 $\mathtt{AC^0[p]}$-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 5.2.1 and 5.2.2.

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

Theorem 3. *A* $\mathtt{TC^0}$*-Frege proof of depth* d *can be effectively* $\mathtt{p}$*-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 5.3.

We also improve the results of Raz and Tzameret [25] to show that Polynomial Calculus with linear transformations can simulate *semantic* Cutting Planes with small coefficients.

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

Theorem 4 is proved in sections 4.1 and 4.2.

1.2 Related Work

Pitassi [22, 23] 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 $\mathtt{VP}$ from $\mathtt{VNP}$. 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 [25] 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 [19]) 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* [24] 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, we formally state all of our results. In section 4.1, we sketch the simulation of syntactic Cutting Planes with bounded coefficients from [25], since it is essential for a significant part of the subsequent discussion. In section 4.2, we extend the simulation to the semantic case, proving Theorem 4. In section 5.1, we prove an analog of the results in section 4.1 over a large enough finite field extension, for use in subsequent sections. In sections 5.2.1, 5.2.2, 5.3, we use techniques from this analog to prove Theorems 2 and 3. Finally in section 5.4, we prove Theorem 1. Technical details of simulations from each of the above sections are contained in the Appendix.

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. Multidimensional 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. $\mathtt{p}$*-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* $\mathtt{p}$*-simulation [24]*
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 $\Delta = \{A_1, \dots, A_m\}$ be a set of unsatisfiable integer linear inequalities in boolean variables $x_1, \dots, x_n$ 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 $B_1, \dots, B_s$ such that B_s is the inequality $0 \geq 1$ and for every $\ell \in \{1, \dots, s\}$ $B_\ell \in \Delta$ 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 $x_1, \dots, x_n$ 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. **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 $2^{\mathtt{poly}(\ell)}$ where ℓ is the length of the proof due to [11].*

Definition 6. *Polynomial Calculus (PC)*
Let $\Gamma = \{P_1, \dots, P_m\}$ be a set of polynomials in variables $\{x_1, \dots, x_n\}$ over a field $\mathbb{F}$ such that the system of equations $P_1 = 0, \dots, P_m = 0$ 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. *Positivestellensatz 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, $\Sigma\Pi\Sigma$-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 $\Sigma\Pi\Sigma$ algebraic circuit.

Definition 8. $\Sigma\Pi\Sigma$-*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* $\Sigma\Pi\Sigma$-*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 $\Sigma\Pi\Sigma$-*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 [25], which is the starting point of our work.

Theorem 0. *[25] $\Sigma\Pi\Sigma$-PC over $\mathbb{Q}$ can simulate syntactic Cutting Planes with size polynomial in n and the coefficient size.*

3 Formal statement of results

We can now restate our results in terms of the proof systems defined in the previous section.

4 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 $\Sigma\Pi\Sigma$-PC*
For a line $A_j \equiv \sum_i a_{ij} x_i \geq b_j$ its translation in $\Sigma\Pi\Sigma$-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 Appendix A.1.

4.1 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. 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 Appendix A.2.

4.2 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*(see Lemma 3), 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 10. *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 Appendix A.3.

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(P_1,P_2)
    C' ← ∅
    for (a,b) ∈ R_y(P_1) × R_z(P_2) 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 10, $(y,z,w) = (a,b,c)$ 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, $\bigcap\limits_{I \in \mathcal{I}_{a,b}} \mathcal{R}_w(I)$ 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 $\mathcal{R}_w(J_{a,b}) = \bigcap\limits_{I \in \mathcal{I}_{a,b}} \mathcal{R}_w(I)$. 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* ; see Lemma 7) would give an equation free of y and z, where w ranges over $\bigcup\limits_{(a,b)} \mathcal{R}_w(J_{a,b})$. Any semantic consequence P_3 must be such that $\mathcal{R}_w(P_3) \supseteq C'$ and hence is easily derived.

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

5.1 Simulating syntactic CP*

We now carry out the simulation in Section 4.1 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 4.1 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 Appendix A.4

5.2 Simulating $\mathtt{AC}^0[\mathtt{q}]$-Frege

5.2.1 Case of $q = p$

For the purpose of this section, we set $d = 9$. We will use the simulation of $\mathtt{AC}^0[\mathtt{p}]$-Frege in [20] 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 $m = O(\mathtt{poly}(\log(\mathtt{n})))$, so that the field we are working over is quasipolynomial sized. Below we describe the proof system of [20] and their simulation of $\mathtt{AC}^0[\mathtt{p}]$-Frege.

The Proof System of Maciel and Pitassi Maciel and Pitassi [20] 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 [20]

initial sequents

1. $\varphi \to \varphi$ for any formula φ
2. $\to \wedge()$; $\vee() \to$
3. $\oplus^p_j() \to$ for $1 \leq j < p$; $\to \oplus^p_0()$
4. $Th_t() \to$
5. $\to Th_0(\varphi_1 \cdots \varphi_k)$ for any $k \geq 0$

structural rules

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

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

permute: $$\frac{\Gamma, \varphi_1, \varphi_2, \Delta \to \Gamma'}{\Gamma, \varphi_2, \varphi_1, \Delta \to \Gamma'} \qquad \frac{\Gamma \to \Gamma', \varphi_1, \varphi_2, \Delta'}{\Gamma \to \Gamma', \varphi_2, \varphi_1, \Delta'}$$

cut rule

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

logical rules

$\neg$: $$\frac{\Gamma \to \varphi, \Delta}{\neg\varphi, \Gamma \to \Delta} \qquad \frac{\varphi, \Gamma \to \Delta}{\Gamma \to \neg\varphi, \Delta}$$

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

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

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

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

$\oplus_i$-left:
$$\frac{\varphi_1, \oplus^p_{i-1}(\varphi_2 \cdots \varphi_k), \Gamma \to \Delta \quad \oplus^p_i(\varphi_2 \cdots \varphi_k), \Gamma \to \varphi_1, \Delta}{\oplus^p_i(\varphi_1, \varphi_2 \cdots \varphi_k), \Gamma \to \Delta}$$

$\oplus_i$-right:
$$\frac{\varphi_1, \Gamma \to \oplus^p_{i-1}(\varphi_2 \cdots \varphi_k), \Delta \quad \Gamma \to \varphi_1, \oplus^p_i(\varphi_2 \cdots \varphi_k), \Delta}{\Gamma \to \oplus^p_i(\varphi_1, \varphi_2 \cdots \varphi_k), \Delta}$$

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

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

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 $\Gamma \to \Delta$, 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.

$tr(x_i) = 1 - x_i$

$tr(\vee(\varphi_1 \cdots \varphi_k)) = \prod_i (tr(\varphi_i))$

$tr(\wedge(\varphi_1 \cdots \varphi_k)) = 1 - \prod_i tr(\neg\varphi_i)$

$tr(\oplus_i^p(\varphi_1 \cdots \varphi_k)) = (\sum_{j=1}^{k} \varphi_j - i)^{p-1}$ for $0 \le i < p$

$tr(Th_t(\varphi_1 \cdots \varphi_k)) = (y - \alpha^t) \cdots (y - \alpha^k)$
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 $t \ge 1$

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

Theorem 6. *[2]*
Any $\mathtt{AC}^0[\mathtt{p}]$ 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 [20]. For an $\mathtt{AC}^0[\mathtt{p}]$ circuit φ, its *flattening* $fl(\varphi)$ is defined as the flat circuit given by the above theorem. Proofs in $\mathtt{AC}^0[\mathtt{p}]$-Frege can be thought of as a list of sequents such that every formula that appears in each of them is an $\mathtt{AC}^0[\mathtt{p}]$ circuit. For a sequent $\varphi_1 \cdots \varphi_k \rightarrow \varphi'_1 \cdots \varphi'_{k'}$ that appears in a $\mathtt{AC}^0[\mathtt{p}]$-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 [20] states the following

Theorem 7. *[20]*
Let $\mathcal{S}$ be a sequent which has a depth d proof in $\mathtt{AC}^0[\mathtt{p}]$-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 Appendix B.1

5.2.2 Case of $q \neq p$

We now extend the simulation of the previous section to show that $\mathtt{AC}^0[\mathtt{q}]$-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 $\mathtt{AC}^0[\mathtt{q}]$-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. The proof for one such rule is shown in Appendix B.2

5.3 Simulating $\mathtt{TC}^0$-Frege

In this section, we show that a $\mathtt{TC}^0$-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 $\mathtt{TC}^0$-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 (see Lemma 14). 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 Appendix C

Lemma 1. *Given the equation*

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

we can derive

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

and vice versa.

Lemma 2. *Given the equation*

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

we can derive

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

and vice versa.

5.3.1 Existence of Feasible Interpolation

Bonet, Pitassi and Raz [4] have shown that $\mathtt{TC}^0$-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*

5.4 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 $d+1$ $\mathtt{TC}^0$ 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 $\xi = \texttt{poly(n)}$ bits [21]. Let $\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_n}$ be the set of n binary numbers, each of length $\xi = \texttt{poly(n)}$, 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 Appendix D

5.4.1 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.

5.4.2 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 $a_1 X_1$ 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*

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

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

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

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 $\mathbf{a}_1 X_1, \cdots, \mathbf{a}_t X_t$ is now defined as $\mathcal{S}^o(\mathbf{a}_1 X_1, \cdots, \mathbf{a}_t X_t) \oplus \mathcal{S}^e(\mathbf{a}_1 X_1, \cdots, \mathbf{a}_n X_t)$.

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

We now define the translation of a line $a_1 X_1 + \cdots + a_n X_k - a_0 \geq 0$ in Cutting Planes/SOS, where $X_1 \ldots X_k$ are monomials.

Definition 18. *Representing an inequality*
Let $P = a_1 X_1 + \cdots + a_k X_k$ be a polynomial where the X_i are monomials. Then the line $P \geq 0$ is represented as

$$\mathcal{S}(\mathbf{a}_1 X_1, \cdots, \mathbf{a}_k X_k)(\xi) = 0$$

and $P = 0$ is represented as

$$\mathcal{S}(\mathbf{a}_1 X_1, \cdots, \mathbf{a}_k X_k) = \mathbf{0}$$

Let $\mathcal{R}(P)$ denote the vector $\mathcal{S}(\mathbf{a}_1 X_1, \cdots, \mathbf{a}_k X_k)$.

5.4.4 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 Appendix D.

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

Lemma 25. *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 27. $\vdash \mathcal{S}(\mathbf{y}_1, \cdots, \mathbf{y}_i) - \mathcal{S}(\mathbf{y}_1, \cdots, \mathbf{y}_{i-1}) \oplus \mathbf{y}_i$

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 29. *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 35. *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 3. *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 $\vdash \mathcal{R}(P) - \big(\mathcal{S}(\mathbf{a}_1x_1, \cdots, \mathbf{a}_nx_n) \oplus (-\mathbf{a}_0)\big)$ by Lemma 27, 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 27 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).

5.4.5 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 41. *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 35 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 36. *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 31. *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 32. *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 36 to derive $\mathcal{R}(P^2)(\xi) = 0$. In the other case, we can use Lemma 31 and Lemma 32 to derive that the sign bit of $\mathcal{R}(-P)$ is zero. We can now use Lemma 36 again to derive $\mathcal{R}(P^2)(\xi) = 0$.

5.4.6 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 $\mathtt{AC^0[p]}$-Frege systems, whether using algebraic proofs or not.

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

1. Find any techique 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-$\Pi\Sigma$-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. 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. A note on the power of threshold circuits. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 580–584. IEEE, 1989.

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

[4] Maria Luisa Bonet, Toniann Pitassi, and Ran Raz. On interpolation and automatization for frege systems. *SIAM Journal on Computing*, 29(6):1939–1967, 2000.

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

[6] Samuel Buss, Leszek Kołodziejczyk, and Konrad Zdanowski. Collapsing modular counting in bounded arithmetic and constant depth propositional proofs. *Transactions of the American Mathematical Society*, 367(11):7517–7563, 2015.

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

[8] Samuel R. Buss, Russell Impagliazzo, Jan Krajícek, Pavel Pudlák, Alexander A. Razborov, and Jirí Sgall. Proof complexity in algebraic systems and bounded depth frege systems with modular counting. *Computational Complexity*, 6(3):256–298, 1997.

[9] Matthew Clegg, Jeffery Edmonds, and Russell Impagliazzo. Using the groebner basis algorithm to find proofs of unsatisfiability. In *Proceedings*

of the twenty-eighth annual ACM symposium on Theory of computing, pages 174–183. ACM, 1996.

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

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

[12] Mikael Goldmann, Johan Håstad, and Alexander A. Razborov. Majority gates VS. general weighted threshold gates. *Computational Complexity*, 2:277–300, 1992.

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

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

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

[16] Joshua A. Grochow and Toniann Pitassi. Circuit complexity, proof complexity, and polynomial identity testing: The ideal proof system. *J. ACM*, 65(6):37:1–37:59, 2018.

[17] Pavel Hrubes and Iddo Tzameret. 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*, pages 41–51, 2009.

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

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

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

[21] Alexis Maciel and Denis Thérien. Threshold circuits of small majority-depth. *Inf. Comput.*, 146(1):55–83, 1998.

[22] Toniann Pitassi. Algebraic propositional proof systems. In *Descriptive Complexity and Finite Models, Proceedings of a DIMACS Workshop 1996, Princeton, New Jersey, USA, January 14-17, 1996*, pages 215–244, 1996.

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

[24] Toniann Pitassi and Rahul Santhanam. Effectively polynomial simulations. In *ICS*, pages 370–382, 2010.

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

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

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

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

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

Appendix A Small-weight Cutting Planes Simulations

Notational Remark In depth-d-PC, we sometimes use "inline" definitions to indicate the new variables y_j introduced. For instance, the equation

$$x_1(x_1 + 1) = 0$$

represents the equations

$$x_1 y_1 = 0$$
$$y_1 = x_1 + 1$$

Thus when we refer to the monomial corresponding to $x_1(x_1 + 1)$, we are referring to $x_1 y_1$.

Though $\Sigma\Pi\Sigma$-PC captures the effect of size reductions due to allowing linear transformations within the proof, it turns out that it is more powerful than required for our simulation in Theorem 1, so we define the tightest restriction of it where we can still do the simulation.

Definition 20. *A Trinomial is a polynomial with at most three monomials*

Definition 21. *Trinomial-$\Pi\Sigma$-PC*
Let $\Gamma = \{P_1, \ldots, P_m\}$ be a set of polynomials over a field $\mathbb{F}$ such that each $P \in \Gamma$ is either an affine form or a trinomial in $\{x_1, \ldots, x_n\}$. Let the system of equations $P_1 = 0, \ldots, P_m = 0$ have no solution. Let $\Gamma' = \{P_1, \ldots, P_m, Q_1, \ldots, Q_k\}$ be a set of polynomials over variables $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_k\}$ such that $Q_1, \ldots, Q_k$ are polynomials of the form $Q_j = y_j - (a_{j0} + \sum_i a_{ij} x_i)$ for some constants $a_{ij} \in \mathbb{F}$. A Trinomial-$\Pi\Sigma$-PC refutation $R_1 \cdots R_s$ of Γ is a Polynomial Calculus refutation of Γ', such that each R_ℓ is either an affine form or a trinomial in $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_k\}$.

Trinomial-$\Pi\Sigma$-PC essentially allows each line in the proof to be a $\Sigma\Pi\Sigma$ circuit in X with the top fan-in bounded by 3. We will measure the size of a Trinomial-$\Pi\Sigma$-PC proof by the number of lines, which is clearly polynomially equivalent to the number of monomials in X, Y. This proof system seems quite restricted, especially since it can no longer trivially simulate Polynomial Calculus unlike $\Sigma\Pi\Sigma$-PC. But surprisingly, the Pigeonhole Principle and Tseitin formulas, for which we have lower bounds for Polynomial Calculus, have small proofs in Trinomial-$\Pi\Sigma$-PC.

A.1 Proof of the Intersection lemma

Here we prove the Intersection lemma and some of its variants that will be used later.

Lemma 4. *"Substitution Lemma"*
Let $R(z-a_1)\cdots(z-a_k) = 0$ and $Rp(z) = 0$ be two equations in a depth-d'-PC refutation, where R is any polynomial and p is a univariate polynomial of degree d in z such that $p(a_i) \neq 0$ for any i. Then, we can derive the equation $R = 0$ in $O(kd|R|)$ lines where $|R|$ is the number of monomials in R.

Proof. Consider the base case of $k = 1$. Starting with $R(z - a_1) = 0$, we can successively derive $Rz^i - Ra_1^i = 0$ for $i \in \{2 \cdots d\}$ by multiplying with the appropriate polynomials in z. This takes $O(d|R|)$ lines in total. Then adding these equations up with the appropriate coefficients we obtain $Rp(z) - Rp(a) = 0$. Since $p(a) \neq 0$ and $Rp(z) = 0$, we have $R = 0$. Now, multiplying every line of the above derivation with $(z - a_2)\cdots(z - a_k)$, we have a derivation of $R(z-a_2)\cdots(z-a_k) = 0$ from $R(z-a_1)\cdots(z-a_k) = 0$ and $Rp(z) = 0$. The lemma now follows by induction over k. $\square$

Lemma 5. *Let $Q(z-a)=0$ and $Q\prod_{i=1}^{k}(z-b_i)=0$ be two equations in Trinomial-$\Pi\Sigma$-PC, where Q is a monomial and $a \neq b_i$ for any i. Then we can derive $Q=0$ in $O(k)$ lines.*

Proof. The proof is by induction on k. The base case, when $k=0$, is trivial. Assume that the lemma is true for some $k-1 \geq 0$. Let $z_1 = z-a$, $z_2 = z-b_1$ and $Q_1 = \prod_{i=2}^{k}(z-b_i)$. The equations are then represented as

$$Qz_1 = 0 \tag{1}$$

$$QQ_1z_2 = 0 \tag{2}$$

$$z_1 = z - a \tag{3}$$

$$z_2 = z - b \tag{4}$$

Multiplying equation (1) by Q_1, we have

$$QQ_1z_1 = 0 \tag{5}$$

Let $c = a - b$. By subtracting (4) from (3) we derive

$$z_1 - z_2 + c = 0 \tag{6}$$

Now multiplying the above equation by the monomial QQ_1, we derive the trinomial

$$QQ_1z_1 - QQ_1z_2 + cQQ_1 = 0$$

But since we already have $QQ_1z_1 = 0$ from (5) and $QQ_1z_2 = 0$ from (2), we obtain

$$cQQ_1 = 0$$

Since $c \neq 0$, we derive $QQ_1 = 0$. Therefore, we now have the equations

$$Q(z-a) = 0$$

$$Q\prod_{i=2}^{k}(z-b_i) = 0$$

The proof of the lemma thus follows from the induction hypothesis. Since it only takes a constant number of lines to go from the case of k to the case of $k-1$, the total number of lines in the derivation is $O(k)$.

□

We now generalize this lemma as follows.

Lemma 6. *"Intersection Lemma"*
Let A and B be two sets of constants in $\mathbb{F}$. Let $\prod_{a\in A}(z-a)=0$ and $\prod_{b\in B}(z-b)=0$ be two equations in Trinomial-$\Pi\Sigma$-PC. Then there is a proof of $\prod_{c\in A\cap B}(z-c)=0$ in Trinomial-$\Pi\Sigma$-PC of length $O(|A\setminus B|\cdot|B\setminus A|)$

Proof. We will prove the lemma by induction over the size of $|A \setminus B|$. The base case when $|A \setminus B| = 0$ trivially follows since $A = A \cap B$.

Now for any two sets A and B such that $|A \setminus B| > 0$, let the equations be labeled as follows

$$\prod_{a \in A} (z - a) = 0 \tag{7}$$

$$\prod_{b \in B} (z - b) = 0 \tag{8}$$

Let $A_0 = A \setminus B$ and $B_0 = B \setminus A$. Choose an element $a_1 \in A_0$. Let $A_1 = A \setminus \{a_1\}$ and $A_2 = A_0 \setminus \{a_1\}$. Let Q_1 be the monomial $\prod_{a \in A_1}(z - a)$ and Q_2 be the monomial $\prod_{a \in A_2}(z - a)$. Then equation (7) can be written as

$$Q_1(z - a_1) = 0 \tag{9}$$

Multiplying (8) by Q_2 we get

$$\prod_{b \in B \cup A_2} (z - b) = 0 \tag{10}$$

Note that there are no squared terms in the monomial since A_2 and B are disjoint. The above equation can be rewritten as

$$\prod_{b \in A_1 \cup B_0} (z - b) = 0 \tag{11}$$

since $A_1 \cup B_0 = B \cup A_2 = (A \cup B) \setminus \{a_0\}$. Note that A_1 and B_0 are also disjoint. Hence we can write the above equation as

$$Q_1 \prod_{b \in B_0} (z - b) = 0 \tag{12}$$

Now since $a_1 \notin B_0$, we can apply Lemma 5 on equations (9) and (12) to get

$$Q_1 = 0$$

i.e.

$$\prod_{a \in A_1} (z - a) = 0 \tag{13}$$

in $O(|B_0|) = O(|B \setminus A|)$ lines.

Now we have two sets of constants A_1 and B with corresponding equations (13) and (8) such that $|A_1 \setminus B| = |A \setminus B| - 1$. Thus the lemma follows by induction. The total number of lines is $O(|A \setminus B| \cdot |B \setminus A|)$.

□

Remark It is easy to see that starting with $Q\prod_{a\in A}(z-a)=0$ and $Q\prod_{b\in B}(z-b)=0$, we can still apply the Intersection Lemma to get $Q\prod_{c\in A\cap B}(z-c)=0$ for any monomial Q.

A.2 Simulating syntactic CP* in Trinomial-ΠΣ-PC over $\mathbb{Q}$

We are now ready to state and prove Theorem 0, which first appeared in [25].

For each possible derivation rule in a Cutting Planes proof, we will now show how to derive in Trinomial-ΠΣ-PC (See Definition 21) the translation of the result of applying the rule on a line or a pair of lines, given their translations.

Simulating Addition For the addition rule, given the translations of two lines $\sum_i a_{ij}x_i \geq b_j$ and $\sum_i a_{ik}x_i \geq b_k$ in CP*, we will derive the translation of their sum $\sum_i (a_{ik}+a_{ij})x_i \geq b_j + b_k$. The following lemma suffices.

Lemma 7. *Simulating addition*
Let $x(x-1)\cdots(x-a)=0$ and $y(y-1)\cdots(y-b)=0$ be two equations in a Trinomial-ΠΣ-PC refutation with $a\geq b$. Then we can derive

$$(x+y)(x+y-1)\cdots(x+y-(a+b))=0$$

using $O(ab)$ lines.

Proof. Let $z=x+y$. We will first derive the range of values z can take when $y=j$, for all $j\in\{0,\ldots,b\}$. Let $x_i=x-i$ for $i\in\{0,\ldots,a\}$, $y_j=y-j$ for $j\in\{0,\ldots,b\}$ and $z_k=z-k$ for $k\in\{0,\ldots,a+b\}$. Also, for $S\subseteq\{0,\ldots,b\}$, let $Y_S=\prod_{j\in\{0,\ldots,b\}\setminus S} y_j$. We denote $Y_{\{j\}}$ simply by Y_j. Then we have

$$z_j = x_0 + y_j$$

Multiplying the above equation by the monomial Y_j, we have

$$z_jY_j - x_0Y_j - y_jY_j = 0$$

Since $y_jY_j=\prod_{j\in\{0\cdots b\}} y_j = 0$, we have

$$z_jY_j - x_0Y_j = 0 \tag{14}$$

It is easy to derive for $i\in\{0\cdots a\}$

$$z_j - z_{j+i} - i = 0$$

Multiplying the above equation by the monomial Y_j, we have

$$z_jY_j - z_{j+i}Y_j - iY_j = 0 \tag{15}$$

Subtracting this from (14) we get

$$z_{j+i}Y_j - x_0Y_j + iY_j = 0 \tag{16}$$

By the definition of x_i we have

$$x_i = x_0 - i$$

Multiplying the above equation by the monomial Y_j, we get

$$x_iY_j - x_0Y_j + iY_j = 0$$

Subtracting the above equation from (16) we get

$$z_{j+1}Y_j - x_iY_j = 0$$

Thus, for all $i \in \{0 \cdots a\}$ we derive

$$z_{j+i}Y_j - x_iY_j = 0$$

From the above $a+1$ equations, we can inductively derive for $i \in \{0 \cdots a\}$

$$z_j \cdots z_{j+i}Y_j - x_0 \cdots x_iY_j = 0$$

as follows. For $i \in \{1 \cdots a\}$, using

$$z_j \cdots z_{j+i-1}Y_j - x_0 \cdots x_{i-1}Y_j = 0$$

we can derive

$$z_j \cdots z_{j+i}Y_j - x_0 \cdots x_{i-1}z_{j+i}Y_j = 0 \tag{17}$$

by multiplying with z_{j+1}. Now multiplying

$$z_{j+i}Y_j - x_iY_j = 0$$

by the monomial $x_0 \cdots x_{i-1}$, we derive

$$x_0 \cdots x_{i-1}z_{j+i}Y_j - x_0 \cdots x_iY_j = 0 \tag{18}$$

Subtracting (18) from (17) we get

$$z_j \cdots z_{j+i}Y_j - x_0 \cdots x_iY_j = 0$$

using $O(j)$ monomials. Therefore, we have

$$z_j \cdots z_{j+a}Y_j - x_0 \cdots x_aY_j = 0 \tag{19}$$

and since $x_0 \cdots x_a = 0$, we derive

$$z_j \cdots z_{j+a} Y_j = 0$$

We derive the above for every $j \in \{0 \cdots b\}$ using a total of $O(ab)$ lines. Multiplying the above line by $\{z_k : 0 \leq k < j\} \cup \{z_k : j + a < k \leq a + b\}$, we have for all $j \in \{0 \cdots b\}$

$$z_0 \cdots z_{a+b} Y_j = 0$$

Now note that the set of monomials $\{Y_j : j \in \{0 \cdots b\}\}$ have no common root. Therefore we can apply the Intersection Lemma repeatedly to derive $z_0 \cdots z_{a+b} = 0$ as follows. Starting with

$$z_0 \cdots z_{a+b} Y_{\{0 \cdots j\}} = 0$$

and

$$z_0 \cdots z_{a+b} Y_{j+1} = 0$$

and applying the Intersection Lemma with $A = \{0 \cdots b\} \setminus \{0 \cdots j\}$ and $B = \{0 \cdots b\} \setminus \{j+1\}$ we get

$$z_0 \cdots z_{a+b} Y_{\{0 \cdots j+1\}} = 0$$

using $O(j)$ lines. Thus using $O(b^2)$ lines we get

$$z_0 \cdots z_{a+b} = 0$$

and the total number of lines is $O(ab + b^2)$. □

Corollary 1. *Given the translations of $\sum_i a_{ij} x_i \geq b_j$ and $\sum_i a_{ik} x_i \geq b_k$, we can derive in Trinomial-$\Pi\Sigma$-PC the translation of $\sum_i (a_{ik} + a_{ij}) x_i \geq b_j + b_k$ in $O((\sum_i a_{ij}^+ - b_j)(\sum_i a_{ik}^+ - b_k))$ lines*

Proof. Use the above lemma for $x = \sum_i a_{ij} x_i - b_j$, $a = \sum_i a_{ij}^+ - b_j$ and $y = \sum_i a_{ik} x_i - b_k$, $b = \sum_i a_{ik}^+ - b_k$. □

Simulating multiplication by a constant We use the following lemma to derive the translation of $c \sum_i c_{ij} x_i \geq c d_j$ in Trinomial-$\Pi\Sigma$-PC from the translation of $\sum_i c_{ij} x_i \geq d_j$

Lemma 8. *Let $(z - a_1) \cdots (z - a_k) = 0$ be an equation in Trinomial-$\Pi\Sigma$-PC. We can derive the equation*

$$(z' - ca_1) \cdots (z' - ca_k) = 0$$

where $z' = cz$ in Trinomial-$\Pi\Sigma$-PC for any $c \in \mathbb{Q}$ in $O(k)$ lines.

Proof. The proof is by induction on k. For $k = 0$, the derivation is trivial. Let $z_i = z - a_i$ and $z'_i = z' - ca_i$ for $i \in \{1 \cdots k\}$. Then, for any $k \geq 1$, we are given the equation

$$z_1 \cdots z_k = 0$$

and we want to derive

$$z'_1 \cdots z'_k = 0$$

Since, $z' = cz$, we get $z'_1 = z' - ca_1 = cz_1$ and thus multiplying with $z_2 \cdots z_k$ we get

$$z'_1 z_2 \cdots z_k - cz_1 \cdots z_k = 0$$

But since $z_1 \cdots z_k = 0$ as above, we get

$$z'_1 z_2 \cdots z_k = 0$$

Now by the induction hypothesis we have a derivation of $z'_2 \cdots z'_k = 0$ from $z_2 \cdots z_k = 0$. By multiplying each step of this derivation by z'_1, we have derived $z'_1 \cdots z'_k = 0$ from $z'_1 z_2 \cdots z_k = 0$.

□

Corollary 2. *Given the translation of $\sum_i c_{ij} x_i \geq d_j$, we can derive the translation of $c \sum_i c_{ij} x_i \geq cd_j$ in Trinomial-$\Pi\Sigma$-PC in $O(\sum_i c^+_{ij} - d_j)$ lines*

Proof. Use the above lemma for $z = \sum_i c_{ij} x_i - d_j$ and
$(a_1 \ \cdots \ a_k) = (0 \ \cdots \ \sum_i c^+_{ij} - d_j)$ □

Simulating division by a constant Given the translation of a line $c \sum_i a_{ij} x_i \geq b_j$ in Cutting Planes for some $c > 0$, we will now derive the translation of $\sum_i a_{ij} x_i \geq \lceil b_j / c \rceil$ by the lemma below. We need the following corollary of Lemma 7

Corollary 3. *Let $z = \sum_i a_{ij} x_i$ be an equation in Trinomial-$\Pi\Sigma$-PC, where x_i are boolean variables. Then we can derive*

$$z\Big(z-1\Big)\cdots\Big(z-(\sum_i a^+_{ij})\Big) = 0$$

in $O((\sum_i a^+_i)^2)$ lines.

Proof. Let $a = \sum_{i=1}^n a^+_{ij}$ and let $b = \sum_{i=1}^{n/2} a^+_{ij}$. Assume that we have derived the equations

$$z_1\Big(z_1-1\Big)\cdots\Big(z_1-(\sum_{i=1}^{n/2} a^+_i)\Big) = 0$$

$$z_2\Big(z_2-1\Big)\cdots\Big(z_2-(\sum_{i=n/2+1}^{n} a_i^+)\Big)=0$$

for $z_1 = \sum_{i=1}^{n/2} a_{ij}x_i$ and $z_2 = \sum_{i=n/2+1}^{n} a_{ij}x_i$. We can use Lemma 7 on the above two equations to derive the required equation in $O(b(a-b))$ lines. Continuing this recursively for the above two lines, the total number of lines $L(a)$ to derive $z\Big(z-1\Big)\cdots\Big(z-(\sum_i a_i^+)\Big)=0$ is given by the recurrence $L(a)=L(b)+L(a-b)+O(b(a-b))$, which gives $L(a)=O(a^2)$ by an easy induction. ☐

Lemma 9. *Simulating Division by a constant*
Let $(cz-b)(cz-(b+1))\cdots(cz-d)=0$ be an equation in Trinomial-$\Pi\Sigma$-PC where $z=\sum_i a_{ij}x_i$ such that x_i are boolean variables, $b<d$ and $c>0$. We can derive

$$(z-\lceil b/c\rceil)(z-(\lceil b/c\rceil+1))\cdots(z-\lfloor d/c\rfloor)=0$$

using $O((\sum_i a_i^+)^2+(\sum_i a_i^+)(d-b))$ lines.

Proof. Using Corollary 3 we can derive the following equation in $O((\sum_i a_i^+)^2)$ lines.

$$z\Big(z-1\Big)\cdots\Big(z-(\sum_i a_{ij}^+)\Big)=0 \tag{20}$$

Now, using Lemma 8 on the equation $(cz-b)(cz-(b+1))\cdots(cz-d)=0$ with the multiplication constant equal to $1/c$, we can derive

$$z(z-b/c)\cdots(z-d/c)=0 \tag{21}$$

Note that the constants in parentheses in the above equation are *rational*, and the smallest integer that appears is $\lceil b/c\rceil$ and the largest integer that appears is $\lfloor d/c\rfloor$. Using the Intersection Lemma with equations (20) and (21), we see that only the integer values are retained from (21) which gives us

$$(z-\lceil b/c\rceil)(z-(\lceil b/c\rceil+1))\cdots(z-\lfloor d/c\rfloor)$$

using $O((\sum_i a_i^+)(d-b))$ lines.

☐

Corollary 4. *Given the translation of a line $c\sum_i a_{ij}x_i\geq b_j$ for some $c>0$, we can derive in Trinomial-$\Pi\Sigma$-PC the translation of $\sum_i a_{ij}x_i\geq\lceil b_j/c\rceil$ in $O(c(\sum_i a_{ij}^+)^2)$ lines*

Proof. Apply the above lemma for $z=\sum_i a_{ij}x_i$. ☐

This completes the simulation of a syntactic CP* proof in Trinomial-$\Pi\Sigma$-PC with the simulation having size polynomial in n and the coefficient size of the original proof.

A.3 Simulating semantic CP* in Trinomial-ΠΣ-PC over $\mathbb{Q}$

Lemma 10. *For every infeasible point $(i, j, k) \in A \times B \times C$, an infeasibility equation of the above form can be derived in $O((\sum_i a_i^+)^2 (\sum_i b_i^+)^2 (\sum_i c_i^+)^2)$ lines*

Proof. We proceed by induction on n. Let $y_\ell = \sum_{i=1}^{\ell} a_i x_i$ and z_ℓ, w_ℓ, A_ℓ, B_ℓ, C_ℓ be defined analogously. For the base case of $n = 1$, the equations defining the grid are $y_1(y_1 - a_1) = 0$, $z_1(z_1 - b_1) = 0$ and $w_1(w_1 - c_1) = 0$. The only feasible points in the grid are $(0, 0, 0)$ and (a_1, b_1, c_1), and thus for every other tuple we will derive an infeasibility equation. We show the derivation for one such tuple $(a_1, 0, 0)$. Starting with

$$y_1 = a_1 x_1$$

$$z_1 = b_1 x_1$$

derive

$$z_1 - b_1 = b_1(x_1 - 1)$$

and multiply by y_1 to derive

$$y_1(z_1 - b_1) = a_1 b_1 x_1 (x_1 - 1) = 0$$

Multiplying the above equation by $(w_1 - c_1)$, we have our required infeasibility equation.

To continue the induction and derive all possible infeasibility equations, we observe that a point (i, j, k) for (y_ℓ, z_ℓ, w_ℓ) is infeasible if and only if the points (i, j, k) and $(i - a_\ell, j - b_\ell, k - c_\ell)$ are infeasible for $(y_{\ell-1}, z_{\ell-1}, w_{\ell-1})$. Therefore, assuming the latter, we derive the former as follows. Given

$$\prod_{\substack{a \in A_{\ell-1} \\ a \neq i}} (y_{\ell-1} - a) \prod_{\substack{b \in B_{\ell-1} \\ b \neq j}} (z_{\ell-1} - b) \prod_{\substack{c \in C_{\ell-1} \\ c \neq k}} (w_{\ell-1} - c) = 0$$

and

$$\prod_{\substack{a \in A_{\ell-1} \\ a \neq i - a_\ell}} (y_{\ell-1} - a) \prod_{\substack{b \in B_{\ell-1} \\ b \neq j - b_\ell}} (z_{\ell-1} - b) \prod_{\substack{c \in C_{\ell-1} \\ c \neq k - c_\ell}} (w_{\ell-1} - c) = 0$$

we will derive

$$\prod_{\substack{a \in A_\ell \\ a \neq i}} (y_\ell - a) \prod_{\substack{b \in B_\ell \\ b \neq j}} (z_\ell - b) \prod_{\substack{c \in C_\ell \\ c \neq k}} (w_\ell - c) = 0$$

Starting with the equations

$$y_\ell = y_{\ell-1} + a_\ell x_\ell$$

$$z_\ell = z_{\ell-1} + b_\ell x_\ell$$

$$w_\ell = w_{\ell-1} + c_\ell x_\ell$$

multiply each by $(x_\ell - 1)$ to derive

$$y_\ell(x_\ell - 1) = y_{\ell-1}(x_\ell - 1)$$
$$z_\ell(x_\ell - 1) = z_{\ell-1}(x_\ell - 1)$$
$$w_\ell(x_\ell - 1) = w_{\ell-1}(x_\ell - 1)$$

From the above equations, it is easy to derive (see Lemma 7)

$$(x_\ell - 1) \prod_{\substack{a \in A_{\ell-1} \\ a \neq i}} (y_\ell - a) \prod_{\substack{b \in B_{\ell-1} \\ b \neq j}} (z_\ell - b) \prod_{\substack{c \in C_{\ell-1} \\ c \neq k}} (w_\ell - c) \tag{22}$$
$$= (x_\ell - 1) \prod_{\substack{a \in A_{\ell-1} \\ a \neq i}} (y_{\ell-1} - a) \prod_{\substack{b \in B_{\ell-1} \\ b \neq j}} (z_{\ell-1} - b) \prod_{\substack{c \in C_{\ell-1} \\ c \neq k}} (w_{\ell-1} - c) \tag{23}$$
$$= 0 \tag{24}$$

Similarly, we derive from the three starting equations

$$y_\ell - a_\ell = y_{\ell-1} + a_\ell(x_\ell - 1)$$
$$z_\ell - b_\ell = z_{\ell-1} + b_\ell(x_\ell - 1)$$
$$w_\ell - c_\ell = w_{\ell-1} + c_\ell(x_\ell - 1)$$

Multiplying by x_ℓ we have

$$(y_\ell - a_\ell)x_\ell = y_{\ell-1}x_\ell$$
$$(z_\ell - b_\ell)x_\ell = z_{\ell-1}x_\ell$$
$$(w_\ell - c_\ell)x_\ell = w_{\ell-1}x_\ell$$

Analogous to the above we can derive

$$x_\ell \prod_{\substack{a \in A_{\ell-1} \\ a \neq i - a_\ell}} (y_\ell - (a + a_\ell)) \prod_{\substack{b \in B_{\ell-1} \\ b \neq j - b_\ell}} (z_\ell - (b + b_\ell)) \prod_{\substack{c \in C_{\ell-1} \\ c \neq k - c_\ell}} (w_\ell - (c + c_\ell)) \tag{25}$$
$$= x_\ell \prod_{\substack{a \in A_{\ell-1} \\ a \neq i - a_\ell}} (y_{\ell-1} - a) \prod_{\substack{b \in B_{\ell-1} \\ b \neq j - b_\ell}} (z_{\ell-1} - b) \prod_{\substack{c \in C_{\ell-1} \\ c \neq k - c_\ell}} (w_{\ell-1} - c) \tag{26}$$
$$= 0 \tag{27}$$

As $A_{\ell-1} \cup \{a + a_\ell : a \in A_{\ell-1}\} \subseteq A_\ell$ (similarly for B_ℓ and C_ℓ), we have from equations (22) and (25)

$$(x_\ell - 1) \prod_{\substack{a \in A_\ell \\ a \neq i}} (y_\ell - a) \prod_{\substack{b \in B_\ell \\ b \neq j}} (z_\ell - b) \prod_{\substack{c \in C_\ell \\ c \neq k}} (w_\ell - c) = 0 \tag{28}$$

$$x_\ell \prod_{\substack{a \in A_\ell \\ a \neq i}} (y_\ell - a) \prod_{\substack{b \in B_\ell \\ b \neq j}} (z_\ell - b) \prod_{\substack{c \in C_\ell \\ c \neq k}} (w_\ell - c) = 0 \tag{29}$$

Adding the above two equations, we derive the required one. □

A.4 Simulating syntactic CP* in depth-5-PC over $\mathbb{F}_{p^m}$

The following lemmas will be largely similar to the ones in Appendix A.2.

Simulating Addition To simulate the addition rule, it suffices to show the following

Lemma 11. *Let A and B be two sets of constants in any field and let $C = \{ab \mid a \in A, b \in B\}$. Let $\prod_{a \in A}(x - a) = 0$ and $\prod_{b \in B}(x - b) = 0$ be two equations in depth-d-PC. Let $z = xy$. Then the equation*

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

can be derived in $O(|A||B|)$ lines.

Proof. Let $A = \{a_i\}, B = \{b_i\}$, $x_i = x - a_i$ and $y_i = y - b_i$. Note that $x_1 \cdots x_{|A|} = 0 = y_1 \cdots y_{|B|}$. Let $X_j = \prod_{i \neq j} x_i$. Starting with

$$z = xy$$

we can derive

$$z = (x - a_j)y + a_j y$$

Now multiplying the above equation by X_j, we have

$$zX_j = x_1 \cdots x_{|A|} y + a_j y X_j = a_j y X_j$$

Subtracting $a_j b_i X_j$ on both sides we can derive for every i the equation

$$(z - a_j b_i) X_j = a_j (y - b_i) X_j$$

Now, similar to Lemma 7, we can derive from the $|B|$ equations above the equation

$$(z - a_j b_1) \cdots (z - a_j b_{|B|}) X_j = a_j y_1 \cdots y_{|B|} X_j = 0$$

Thus for every j we have the equation

$$(z - a_j b_1) \cdots (z - a_j b_{|B|}) X_j = 0$$

Multiplying each of the above $|A|$ equations with the missing terms, we can obtain for every j,

$$\prod_{c \in C} (z - c) X_j = 0$$

Using the Intersection Lemma inductively as in Lemma 7, we obtain the required equation. $\square$

Corollary 5. *Given the translations of $\sum_i a_{ij} x_i \geq b_j$ and $\sum_i a_{ik} x_i \geq b_k$ in depth-d-PC over F_{p^m}, we can derive the translation of $\sum_i (a_{ik} + a_{ij}) x_i \geq b_j + b_k$ in $O((\sum_i a_{ij} - b_j)(\sum_i a_{ik} - b_k))$ lines*

Proof. Use the above lemma for $y_1 = \prod_i ((\alpha^{a_{ij}} - 1) x_i + 1)$, $y_2 = \prod_i ((\alpha^{a_{ik}} - 1) x_i + 1)$, $A = \{\alpha^{b_j}, \alpha^{b_j + 1} \cdots \alpha^{\sum_i a_{ij}^+}\}$, $B = \{\alpha^{b_k}, \alpha^{b_k + 1} \cdots \alpha^{\sum_i a_{ik}^+}\}$ $\square$

Simulating Multiplication

Lemma 12. *Let A be a set of constants in any field and let c be a positive integer. Let $A^c = \{a^c \mid a \in A\}$. Let $\prod_{a \in A} (x - a) = 0$ be an equation in the depth-d-PC. Then we can derive the equation*

$$\prod_{a \in A^c} (x^c - a) = 0$$

in $O(|A|)$ lines.

Proof. Let $x_i = x - a_i$ and $x_i' = x_i^c$. Then the given equation becomes $x_1 \cdots x_{|A|} = 0$, and we want to derive $x_1' \cdots x_{|A|}' = 0$. The proof is by induction on $|A|$. If $|A| = 0$ then we have nothing to prove. Assume that the statement is true for $|A| \leq k - 1$ for some $k \geq 1$. Consider an expression of the form $\prod_{a \in A} (x - a) = 0$, where $|A| = k$. If $|A^c| < k$, then clearly there exists a set $A_1 \subset A$ such that $A_1^c = A^c$, and the required equation follows from the induction hypothesis. If $|A^c| = k$, from the given equation, it is easy to derive

$$x x_2 \cdots x_k - a_1 x_2 \cdots x_k = 0$$

Multiplying the above equation with x, we have

$$x^2 x_2 \cdots x_k - a_1 x x_2 \cdots x_k = 0$$

Adding a_1 times the former equation to the latter, we have

$$x^2 x_2 \cdots x_k - a_1^2 x_2 \cdots x_k = 0$$

Proceeding in a similar way, we can derive

$$x^c x_2 \cdots x_k - a_1^c x_2 \cdots x_k = 0$$

or equivalently

$$x_1' x_2 \cdots x_k = 0$$

Now by the induction hypothesis, we have a proof of $x_2' \cdots x_k' = 0$ from $x_2 \cdots x_k = 0$. Multiplying each line of the proof by x_1' we arrive at a proof of the required equation. □

Corollary 6. *Given the translation of $\sum_i a_{ij} x_i \geq b_j$ in depth-d-PC over F_{p^m} and an integer $c < p^m - 1$, we can derive the translation of $\sum_i c a_{ij} x_i \geq c b_j$ in $O((\sum_i a_{ij} - b_j))$ lines*

Proof. Use the above lemma for $y = \prod_i ((\alpha^{a_{ij}} - 1) x_i + 1)$, $A = \{\alpha^{b_j}, \alpha^{b_j + 1} \cdots \alpha^{\sum_i a_{ij}^+}\}$ □

Note that previous two lemmas hold over any field. For the following lemma, we will use the fact that we are working over F_{p^m} where $s^2 < p^m - 1$.

Simulating Division The proof of the following corollary is analogous to Corollary 3.

Corollary 7. *Let $x = \prod_i ((\alpha^{b_{ij}} - 1) x_i + 1)$ be a variable where x_i are boolean. We can derive*

$$(x-1)(x-\alpha) \cdots (x - \alpha^{\sum_i b_{ij}^+}) = 0$$

in $O((\sum_i b_{ij}^+)^2)$ lines

Lemma 13. *Let $(x^c - \alpha^{c a_1}) \cdots (x^c - \alpha^{c a_k}) = 0$ be an equation in depth-d-PC over $\mathbb{F}_{p^m}$, where a_i are distinct and x is of the form $\prod_i ((\alpha^{b_{ij}} - 1) x_i + 1)$ where x_i are boolean. There is a proof of the equation*

$$(x - \alpha^{a_1}) \cdots (x - \alpha^{a_k}) = 0$$

in $O((\sum_i a_i^+)^2)$ lines

Proof. Using Corollary 7, we can derive

$$(x-1)(x-\alpha) \cdots (x - \alpha^{\sum_i b_{ij}^+}) = 0 \tag{30}$$

in $O((\sum_i b_{ij}^+)^2)$ lines. Since $\sum_i |b_{ij}| < s$, any term $(x - \alpha^b)$ that appears in the above equation is such that $b \in [0, s]$ or $b \in [p^m - 1 - s, p^m - 2]$.

The proof is by induction on k. Consider the case of $k = 1$, when we have the equation $x^c - \alpha^{ca_1} = 0$ where $a_1 \leq s$ without loss of generality. If $c \nmid p^m - 1$, then it has a unique root α^{a_1}. If $c \mid p^m - 1$, then the roots are of the form $\alpha^{a_i + j(p^m-1)/c}$ for $j \in \{0 \cdots c-1\}$. But since $2s^2 < p^m - 1$,

$$c \leq s < (p^m - 1)/2s \leq (p^m - 1)/2c \tag{31}$$

Therefore any root α^b such that $b \neq a_1$ is such that $b \geq a_i + (p^m - 1)/c > s$. Also, we have

$$\begin{aligned} b &\leq a_i + (p^m - 1)(c-1)/c \\ &= p^m - 1 - ((p^m - 1)/c - a_i) \\ &< p^m - 1 - ((p^m - 1)/c - s) \\ &< p^m - 1 - s \end{aligned}$$

where the last inequality is due to (31). Therefore the only root α^b to the equation $x^c - \alpha^{ca_1} = 0$ such that $b \in [0, s]$ or $b \in [p^m - 1 - s, p^m - 2]$ is α^{a_1}. Starting with the equation $x^c - \alpha^{ca_1} = 0$ it is easy to derive

$$(x - \alpha^{a_1})Q(x) = 0 \tag{32}$$

where $Q(x) = x^{c-1} + \alpha x^{c-2} + \cdots + \alpha^{c-1}$, just by expanding the above equation into its monomials. Now by our discussion above, for any term $(x - \alpha^b)$ that appears in the equation (30), $Q(\alpha^b) \neq 0$. Therefore, using the Substitution lemma with equations (30) and (32) we derive $x - \alpha^{a_1} = 0$ if this term appears in (30), else we derive $1 = 0$. Therefore, this gives a derivation of $x - \alpha^{a_1} = 0$ from the equation $x^c - \alpha^{ca_1} = 0$.

For the induction step, by multiplying every step in the above derivation with
$(x^c - \alpha^{ca_2}) \cdots (x^c - \alpha^{ca_k})$, we obtain a derivation of

$$(x - \alpha^{a_1})(x^c - \alpha^{ca_2}) \cdots (x^c - \alpha^{ca_k}) = 0$$

from

$$(x^c - \alpha^{ca_1}) \cdots (x^c - \alpha^{ca_k}) = 0$$

The lemma now follows by induction.

□

Corollary 8. *Given the translation of $c \sum_i a_{ij} x_i \geq b_j$ in depth-d-PC over F_{p^m} for an integer $c < p^m - 1$, we can derive the translation of $\sum_i a_{ij} x_i \geq \lceil b_j / c \rceil$ in $O((c \sum_i a_{ij}^+)^2)$ lines*

Proof. Let the equation

$$(y^c - \alpha^{b_j}) \cdots (y^c - \alpha^{c \sum_i a_{ij}^+}) = 0 \tag{33}$$

be obtained from the translation of $c \sum_i a_{ij} x_i \geq b_j$, where $y = \prod_i ((\alpha^{a_{ij}} - 1)x_i + 1)$. We first use Corollary 7 to derive

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

in $(\sum_i a_{ij}^+)^2$ lines. Using Lemma 12 on the above equation, we get

$$(y^c - 1)(y^c - \alpha^c) \cdots (y^c - \alpha^{c \sum_i a_{ij}^+}) = 0 \tag{34}$$

in $\sum_i a_{ij}^+$ lines. Using the Intersection Lemma on equations (33) and (34), we get

$$(y^c - \alpha^{c \lceil b_j / c \rceil}) \cdots (y^c - \alpha^{c \sum_i a_{ij}^+}) = 0$$

We now use the previous lemma to derive

$$(y - \alpha^{\lceil b_j / c \rceil}) \cdots (y - \alpha^{\sum_i a_{ij}^+}) = 0$$

which is the required equation.

□

This completes the proof of Theorem 5.

Appendix B Simulating $\mathtt{AC}^0[\mathtt{q}]$-Frege in depth-9-PC over $\mathbb{F}_{p^m}$

B.1 Case of $q = p$

Simulating Initial sequents

Here we will show how to derive translations of the initial sequents from $x_i(1 - x_i) = 0$.

Lemma 14. *Let φ be any formula of depth three which only contains the $\oplus_i^p$, $\neg$, $\wedge$ and $\vee$ connectives. Then the equation $tr(\varphi)(1 - tr(\varphi)) = 0$ can be derived from $x_i(x_i - 1) = 0$ in depth-d-PC*

Proof. Easily follows from repeated application of Lemmas 7, 11 and 12 at each level. □

Lemma 15. *The translation of the initial sequent $\varphi \to \varphi$ can be derived from $x_i(x_i - 1) = 0$ in depth-d-PC for any flat circuit φ*

Proof. If φ is a flat circuit without threshold gates, this follows by Lemma 14 since the translation of the sequent $\varphi \to \varphi$ is simply $tr(\varphi)(1-tr(\varphi)) = 0$. If φ contains a top threshold gate, the translation of the given sequent states that a variable y such that $y = \prod_{i=1}^{k}((\alpha - 1)tr(\neg\varphi_i) + 1)$ satisfies $(y-1)\cdots(y-\alpha^k) = 0$, where φ_i are formulas without threshold gates. Thus we can derive $tr(\neg\varphi_i)(1 - tr(\neg\varphi_i)) = 0$ as in Lemma 14 and then use Lemma 7 to derive $(y-1)\cdots(y-\alpha^k) = 0$. □

The initial sequents 2,3 and 4 are dummies and do not require translating. The initial sequent 5 can be derived using Lemma 7 since in a flat proof each of the inputs to the threshold connective do not contain threshold connectives.

Simulating structural rules

The simulation of the weakening rule just involves multiplying the given equation by the translation of the new formula φ that appears. The permutation rule is trivial since the translation of a sequent is invariant under application of the permutation rule. To simulate the contraction rule, we need to show that for every formula φ, we can derive from $(tr(\varphi))^2 = 0$ the equation $tr(\varphi) = 0$. When φ is a formula which does not involve a threshold connective, this is can be done by using Lemma 14. When φ is a flat circuit with a threshold gate at the top, the following lemma suffices.

Lemma 16. *Let $(y-\alpha^{a_1})^2\cdots(y-\alpha^{a_{k'}})^2 = 0$ be an equation in depth-d-PC where a_i are distinct integers less than $p^m - 1$ and $y = \prod_{i=1}^{k}((\alpha - 1)tr(\neg\varphi_i) + 1)$ such that φ_i are flat formulas with no threshold gates. The equation $(y-\alpha^{a_1})\cdots(y-\alpha^{a_{k'}}) = 0$ can be derived in $O(\max(k', k^2))$ lines.*

Proof. The proof is by induction on k'. The case of $k' = 0$ is trivial. Using Lemma 7 we can derive the range of values of the variable y, i.e. an equation of the form

$$(y-1)\cdots(y-\alpha^k) = 0 \tag{35}$$

Let $Q = (y-\alpha^{a_1})(y-\alpha^{a_2})^2\cdots(y-\alpha^{a_{k'}})^2$ and $Q_1 = (y-\alpha^{a_2})^2\cdots(y-\alpha^{a_{k'}})^2$. Then the given equation can be written as

$$Q(y-\alpha^{a_1}) = 0 \tag{36}$$

Multiplying equation (35) with Q if it does not contain the term $(y-\alpha^{a_1})$, else multiplying it with Q_1, we arrive at

$$Q \prod_{1\le i\le k\ ,\ i\ne a_1} (y-\alpha^i)$$

Using Lemma 5 with equations (35) and (36), we get $Q = 0$. The lemma now follows by induction since assuming there is a derivation of

$(y-\alpha^{a_2})\cdots(y-\alpha^{a_{k'}})=0$ from $(y-\alpha^{a_2})^2\cdots(y-\alpha^{a_{k'}})^2$, this derivation can be multiplied by $(y-\alpha^{a_1})=0$ to get the required equation from $Q=0$. $\square$

Simulating the cut rule

Let $Q=tr(\neg\Gamma)tr(\Delta)$ and $Q'=tr(\neg\Gamma')tr(\Delta')$. Let $y=tr(\varphi)$ if φ does not contain threshold gates, else let $y=\prod_{i=1}^{k}((\alpha-1)tr(\neg\varphi_i)+1)$ where $\varphi=Th_t(\varphi_1\cdots\varphi_k)$. Then the cut rule can be translated to the following statement

Lemma 17. *Given the equations $Q(y-a_1)\cdots(y-a_k)=0$ and $Q'(y-b_1)\cdots(y-b_{k'})=0$ where $a_1\cdots a_k$ and $b_1\cdots b_{k'}$ are disjoint sets of constants from the field, derive $QQ'=0$*

Proof. Multiply the first equation by Q' and the second equation by Q, and use the contraction rule to make sure the resulting equations are square free. Then required equation now follows easily from the Intersection Lemma. $\square$

Simulating $\wedge$,$\vee$, $\oplus_i^p$ and $\neg$ rules

The rules for $\neg$, $\wedge$-left and $\vee$-right are trivially simulated since the translation remains invariant. For the $\wedge$-right and $\vee$-left, the simulation reduces to the following lemma, where $Q=tr(\neg\Gamma)tr(\Delta)$.

Lemma 18. *Given the equations $Qy_1=0$ and $Qy=0$ where y_1 and y take boolean values, derive the equation $Qyy_1=0$*

Proof. Follows from Lemma 7 $\square$

For the $\wedge$-right rule, the above lemma can be instantiated with $y_1=tr(\varphi_1)$ and $y=tr(\wedge(\varphi_2\cdots\varphi_k))$. Since $\wedge(\varphi_1\cdots\varphi_k)$ is being derived, each of the formulas φ_i must be free of threshold gates. Thus the fact that y and y_1 are boolean is easily derived from Lemma 14. A similar simulation works for the $\vee$-left rule.

The simulation for $\oplus_i^p$ gates is analogous to the above. Let $Q=tr(\neg\Gamma)tr(\Delta)$, and $x_i=tr(\varphi_i)$. The $\oplus_1^p$-left rule then translates to the following lemma. The simulations for the other $\oplus_i^p$ rules are similar.

Lemma 19. *Given the equations*

$$x_1(1-z_2^{p-1})=0$$

and

$$(1-x_1)(1-(1-z_2)^{p-1})=0$$

derive $(1-(1-z_1)^{p-1})=0$, where $z_1=x_1+\cdots x_n$, $z_2=x_2+\cdots x_n$ and x_i are boolean variables.

Proof. Starting with the equation

$$z_1 = x_1 + z_2$$

Multiply by $(1 - x_1)$ on both sides and subtract $(1 - x_1)$ to get

$$(z_1 - 1)(1 - x_1) = x_1(1 - x_1) + (z_2 - 1)(1 - x_1) = (z_2 - 1)(1 - x_1)$$

Now, we can raise both sides of the equation to the exponent $p - 1$, and use the fact that $(1 - x_1)^{p-1} = (1 - x_1)$ (which is easily derived using Lemma 12) to get

$$(z_1 - 1)^{p-1}(1 - x_1) = (z_2 - 1)^{p-1}(1 - x_1)$$

But since from the second equation of our hypothesis, $(z_2 - 1)^{p-1}(1 - x_1) = (1 - x_1)$ and thus

$$(1 - (z_1 - 1)^{p-1})(1 - x_1) \tag{37}$$

Now consider the equation

$$z_1 - 1 = x - 1 + z_2$$

obtained by subtracting one from $z_1 = x_1 + z_2$

Multiplying by x on both sides, we get

$$(z_1 - 1)x_1 = x(x - 1) + z_2 x = z_2 x$$

Again, raising to the exponent $p - 1$ and noting that $x_1^{p-1} = x_1$ and $z_2^{p-1} x_1 = x_1$ we have

$$(z_1 - 1)^{p-1} x_1 = z_2^{p-1} x_1 = x_1$$

and thus

$$(1 - (z_1 - 1)^{p-1})x_1 = 0$$

Adding equation (37) to the above we get the required equation

□

Simulating Th_t rules

Let $Q = tr(\neg\Gamma)tr(\Delta)$, and $x_i = tr(\neg\varphi_i)$. The Th_t-left rule translates to the following lemma. The case of Th_t-right is similar.

Lemma 20. *Given the equations*

$$(z_2 - 1) \cdots (z_2 - \alpha^{t+1}) = 0$$

and

$$x_1(z_2-1)\cdots(z_2-\alpha^t)=0$$

derive

$$(z_1-1)\cdots(z_1-\alpha^{t+1})=0$$

where $z_1=\prod_{i=1}^k((\alpha-1)x_i+1)$, $z_2=\prod_{i=2}^k((\alpha-1)x_i+1)$ *and* x_i *are boolean variables.*

Proof. It is easy to derive the equation

$$z_1=(\alpha x_1+1-x_1)z_2$$

Multiplying the above equation with $(1-x_1)$ we get

$$z_1(1-x_1)=(1-x_1)^2z_2=(1-x_1)z_2$$

since x_1 is boolean. Subtracting $\alpha^i(1-x_1)$ on both sides we get

$$(z_1-\alpha^i)(1-x_1)=(z_2-\alpha^i)(1-x_1)$$

for every i in $\{0\cdots t+1\}$. From these $t+1$ equations it is easy to derive (see Lemma 7)

$$(z_1-1)\cdots(z_1-\alpha^{t+1})(1-x_1)=(z_2-1)\cdots(z_2-\alpha^{t+1})(1-x_1)=0 \quad (38)$$

Multiplying the equation $z_1=(\alpha x_1+1-x_1)z_2$ with x_1 we get

$$z_1x_1=\alpha x_1^2z_2=\alpha x_1z_2$$

Again, subtracting $\alpha^{i+1}x_1$ we get

$$(z_1-\alpha^{i+1})x_1=(z_2-\alpha^i)x_1$$

for every i in $\{0\cdots t\}$. Once again, we combine them to derive

$$(z_1-\alpha)\cdots(z_1-\alpha^{t+1})x_1=(z_2-1)\cdots(z_2-\alpha^t)x_1=0$$

Multiplying the above equation with z_1-1 and adding it to equation (38), we get the required equation. □

This completes the simulation of flat proofs in depth-d-PC.

B.2 Case of $q \neq p$

Lemma 21. *Given the equations*

$$x_1(1 - (y_2 - 1)^{p^m - 1}) = 0$$

and

$$(1 - x_1)(1 - (y_2 - \alpha^r)^{p^m - 1}) = 0$$

derive

$$(1 - (y_1 - \alpha^r)^{p^m - 1}) = 0$$

where $y_1 = \prod_{i=1}^{k}((\alpha^r - 1)x_i + 1)$, $y_2 = \prod_{i=2}^{k}((\alpha^r - 1)x_i + 1)$ *and* x_i *are boolean variables*

Proof. It is easy to derive

$$y_1 = (\alpha^r x_1 + 1 - x_1)y_2$$

Multiplying the above equation with x_1 we have

$$y_1 x_1 = \alpha^r y_2 x_1^2 = \alpha^r y_2 x_1$$

since x_1 is boolean. By subtracting $\alpha^r x_1$ we can now derive

$$(y_1 - \alpha^r)x_1 = \alpha^r x_1(y_2 - 1)$$

Raising the above equation to the power $p^m - 1$, we get

$$(y_1 - \alpha^r)^{p^m - 1} x_1 = x_1(y_2 - 1)^{p^m - 1}$$

since x_1 is boolean. Subtracting the above equation from x_1, we get

$$(1 - (y_1 - \alpha^r)^{p^m - 1})x_1 = (1 - (y_2 - 1)^{p^m - 1})x_1 = 0 \tag{39}$$

By multiplying with $1 - x_1$ we can derive from $y_1 = (\alpha^r x_1 + 1 - x_1)y_2$ the equation

$$y_1(x_1 - 1) = y_2(x_1 - 1)$$

Carrying out a derivation similar to the above, we get

$$(1 - (y_1 - \alpha^r)^{p^m - 1})(x_1 - 1) = (1 - (y_2 - \alpha^r)^{p^m - 1})(x_1 - 1) = 0 \tag{40}$$

Adding equations (39) and (40) we get the required equation. $\square$

Appendix C Simulating $\mathtt{TC}^0$-Frege in depth-d-PC over $\mathbb{F}_{p^m}$

Lemma 22. *Given the equation*

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

we can derive

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

and vice versa.

Proof. In the forward direction, the required equation is easily derived by repeated application of the contraction rule. The other direction is trivial. □

Lemma 23. *Given the equation*

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

we can derive

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

and vice versa.

Proof. In the forward direction, since y is a threshold gate with k arguments, we can derive

$$(y-1)\cdots(y-\alpha^k)=0$$

and thus

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

But since we have $\big((y-\alpha^t)\cdots(y-\alpha^k)\big)^{p^m-1}=1$ from the given equation, we get

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

Using the contraction rule repeatedly gives the required equation.

In the reverse direction, Let $y_1=\big((y-\alpha^t)\cdots(y-\alpha^k)\big)^{p^m-1}$. Then as mentioned earlier, we can derive using Lemma 14

$$y_1(1-y_1)=0$$

Using the contraction rule on the above equation, we get

$$(y-\alpha^t)\cdots(y-\alpha^k)(1-y_1)=0 \tag{41}$$

Multiplying the given equation $(y-1)\cdots(y-\alpha^{t-1})=0$ by $(1-y_1)$ and using the Intersection Lemma with equation (41), we get $1-y_1=0$, which is the required equation. □

Appendix D Dealing with large coefficients

D.1 Properties of addition

In this section we derive some basic properties of addition.

The following lemma shows that our system can prove the associativity of $\oplus$.

Lemma 24. *For bits y, z, w, let $H(y,z) := y \wedge z$ and let $H(y,z,w) := (y \wedge z) \vee (z \wedge w) \vee (w \wedge y)$ which is one if and only if $y + z + w \geq 2$. $H(.)$ denotes the carry bit generated by adding together up to three bits. The following are easily proved since they involve only a constant number of variables.*

$$\vdash H(y,z,w) - H(y, z \oplus w) \oplus H(z,w) \tag{42}$$

$$z_1 + w_1 - (z_2 + w_2) \vdash H(y, z_1, w_1) - H(y, z_2, w_2) \tag{43}$$

$$\vdash H(H(y, z \oplus w), H(z,w)) \tag{44}$$

If c_i are carry bits in $\mathbf{y} \oplus \mathbf{z}$, then

$$\vdash c_{i+1} - H(y_i, z_i, c_i) \tag{45}$$

For bits a, b, c, d, e,

$$\vdash H(a,b,c) + H(a \oplus b \oplus c, d, e) - H(a,b,d) - H(a \oplus b \oplus d, c, e) \tag{46}$$

Lemma 25. *For any three bit vectors $\mathbf{y}$, $\mathbf{z}$ and $\mathbf{w}$*

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

Proof. Let $\mathbf{y}_{left} := (\mathbf{y} \oplus \mathbf{z}) \oplus \mathbf{w}$ and $\mathbf{y}_{right} := \mathbf{y} \oplus (\mathbf{z} \oplus \mathbf{w})$. Let $d_i^{\mathbf{y},\mathbf{z}}$ be the carry bit to the i^{th} position in $\mathbf{y} \oplus \mathbf{z}$. Let $d_i^{\mathbf{w}}$ be the carry bit to the i^{th} position in $(\mathbf{y} \oplus \mathbf{z}) \oplus \mathbf{w}$. Similarly define $d_i^{\mathbf{z},\mathbf{w}}$ and $d_i^{\mathbf{y}}$. We will derive inductively for every i

$$\vdash d_i^{\mathbf{y},\mathbf{z}} + d_i^{\mathbf{w}} - (d_i^{\mathbf{z},\mathbf{w}} + d_i^{\mathbf{y}}) \tag{47}$$

$$\vdash \mathbf{y}_{left}(i) - \mathbf{y}_{right}(i)$$

This is easily derived for $i = 1$. Suppose for some $i \geq 1$ the above lines have been derived.

By (45) of Lemma 24, we derive

$$\vdash d_{i+1}^{\mathbf{y},\mathbf{z}} - H(\mathbf{y}(i), \mathbf{z}(i), d_i^{\mathbf{y},\mathbf{z}})$$

$$\vdash d^{\mathbf{w}}_{i+1} - H(\mathbf{y}(i) \oplus \mathbf{z}(i) \oplus d^{\mathbf{y},\mathbf{z}}_i, \mathbf{w}(i), d^{\mathbf{w}}_i)$$

since $\mathbf{y} \oplus \mathbf{z}(i) = \mathbf{y}(i) \oplus \mathbf{z}(i) \oplus d^{\mathbf{y},\mathbf{z}}_i$. Adding these lines we get

$$\vdash d^{\mathbf{y},\mathbf{z}}_{i+1} + d^{\mathbf{w}}_{i+1} - \big(H(\mathbf{y}(i), \mathbf{z}(i), d^{\mathbf{y},\mathbf{z}}_i) + H(\mathbf{y}(i) \oplus \mathbf{z}(i) \oplus d^{\mathbf{y},\mathbf{z}}_i, \mathbf{w}(i), d^{\mathbf{w}}_i)\big)$$

Using (46) of Lemma 24, we make the derivation

$$\begin{aligned}\vdash H(\mathbf{y}(i), \mathbf{z}(i), d^{\mathbf{y},\mathbf{z}}_i) + H(\mathbf{y}(i) \oplus \mathbf{z}(i) \oplus d^{\mathbf{y},\mathbf{z}}_i, \mathbf{w}(i), d^{\mathbf{w}}_i)&\\ - \big(H(\mathbf{y}(i), \mathbf{z}(i), \mathbf{w}(i)) + H(\mathbf{y}(i) \oplus \mathbf{z}(i) \oplus \mathbf{w}(i), d^{\mathbf{y},\mathbf{z}}_i, d^{\mathbf{w}}_i)\big)&\end{aligned}$$

Adding this to the line above, we get

$$\begin{aligned}\vdash d^{\mathbf{y},\mathbf{z}}_{i+1} + d^{\mathbf{w}}_{i+1}&\\ - \big(H(\mathbf{y}(i), \mathbf{z}(i), \mathbf{w}(i)) + H(\mathbf{y}(i) \oplus \mathbf{z}(i) \oplus \mathbf{w}(i), d^{\mathbf{y},\mathbf{z}}_i, d^{\mathbf{w}}_i)\big)&\end{aligned}$$

In a similar fashion, we make the derivation

$$\begin{aligned}\vdash d^{\mathbf{z},\mathbf{w}}_{i+1} + d^{\mathbf{y}}_{i+1}&\\ - \big(H(\mathbf{y}(i), \mathbf{z}(i), \mathbf{w}(i)) + H(\mathbf{y}(i) \oplus \mathbf{z}(i) \oplus \mathbf{w}(i), d^{\mathbf{z},\mathbf{w}}_i, d^{\mathbf{y}}_i)\big)&\end{aligned}$$

Now, using our induction hypothesis (47) and (43) of Lemma 24, we derive

$$\vdash H(\mathbf{y}(i) \oplus \mathbf{z}(i) \oplus \mathbf{w}(i), d^{\mathbf{y},\mathbf{z}}_i, d^{\mathbf{w}}_i) - H(\mathbf{y}(i) \oplus \mathbf{z}(i) \oplus \mathbf{w}(i), d^{\mathbf{z},\mathbf{w}}_i, d^{\mathbf{y}}_i)$$

The derivation

$$\vdash d^{\mathbf{y},\mathbf{z}}_{i+1} + d^{\mathbf{w}}_{i+1} - (d^{\mathbf{z},\mathbf{w}}_{i+1} + d^{\mathbf{y}}_{i+1})$$

is now easily obtained from the three previous lines.

To derive $\mathbf{y}_{left}(i+1) = \mathbf{y}_{right}(i+1)$, we first make the following derivation

$$d^{\mathbf{y},\mathbf{z}}_{i+1} + d^{\mathbf{w}}_{i+1} - (d^{\mathbf{z},\mathbf{w}}_{i+1} + d^{\mathbf{y}}_{i+1}) \vdash d^{\mathbf{y},\mathbf{z}}_{i+1} \oplus d^{\mathbf{w}}_{i+1} - (d^{\mathbf{z},\mathbf{w}}_{i+1} \oplus d^{\mathbf{y}}_{i+1})$$

since this involves only a constant number of boolean variables. Now, by definition, $\mathbf{y}_{left}(i+1) = \mathbf{y}(i+1) \oplus \mathbf{z}(i+1) \oplus \mathbf{w}(i+1) \oplus d^{\mathbf{y},\mathbf{z}}_{i+1} \oplus d^{\mathbf{w}}_{i+1}$ and by the above two lines this is equal to $\mathbf{y}(i+1) \oplus \mathbf{z}(i+1) \oplus \mathbf{w}(i+1) \oplus d^{\mathbf{z},\mathbf{w}}_{i+1} \oplus d^{\mathbf{y}}_{i+1}$, which is equal to $\mathbf{y}_{right}(i+1)$.

□

The following lemmas show that the addition operations $\mathcal{S}$ and $\oplus$ can be used interchangeably.

Lemma 26. *For $i \leq n$,*

$$\vdash \mathcal{S}^o(\mathbf{y}_1 \cdots \mathbf{y}_{i-1}) \oplus \mathbf{y}^o_i - \mathcal{S}^o(\mathbf{y}_1 \cdots \mathbf{y}_i)$$

$$\vdash \mathcal{S}^e(\mathbf{y}_1 \cdots \mathbf{y}_{i-1}) \oplus \mathbf{y}^e_i - \mathcal{S}^e(\mathbf{y}_1 \cdots \mathbf{y}_i)$$

Proof. We are going to prove the statement block wise. For odd j, let $w_j = \sum_{k=1}^{i-1}[L_j(\mathbf{y}_k^o)]$. Note that the pair of blocks $(j, j+1)$ in $\mathcal{S}^o(\mathbf{y}_1 \cdots \mathbf{y}_i)$ only depend on the corresponding pair of blocks in $\mathcal{S}^o(\mathbf{y}_1 \cdots \mathbf{y}_{i-1})$ and $L_j(\mathbf{y}_i^o)$. Therefore, restricted to the blocks $(j, j+1)$, the statement of the lemma just depends on w_j and $L_j(\mathbf{y}_i^o)$. Since w_j only takes on ξ_0^2 values and $L_j(\mathbf{y}_i^o)$ only takes on ξ_0 values, there is a polynomial sized proof by completeness. □

Lemma 27. $\vdash \mathcal{S}(\mathbf{y}_1 \cdots \mathbf{y}_i) - \mathcal{S}(\mathbf{y}_1 \cdots \mathbf{y}_{i-1}) \oplus \mathbf{y}_i$

Proof. Since $\mathcal{S}(\mathbf{y}_1 \cdots \mathbf{y}_{i-1}) = \mathcal{S}^e(\mathbf{y}_1 \cdots \mathbf{y}_{i-1}) \oplus \mathcal{S}^o(\mathbf{y}_1 \cdots \mathbf{y}_{i-1})$ and $\mathbf{y}_i = \mathbf{y}_i^e \oplus \mathbf{y}_i^o$ by definition, we have

$$\vdash \mathcal{S}(\mathbf{y}_1 \cdots \mathbf{y}_{i-1}) \oplus \mathbf{y}_i - \mathcal{S}^e(\mathbf{y}_1 \cdots \mathbf{y}_{i-1}) \oplus \mathcal{S}^o(\mathbf{y}_1 \cdots \mathbf{y}_{i-1}) \oplus \mathbf{y}_i^e \oplus \mathbf{y}_i^o$$

From Lemma 25, we have

$$\vdash \mathcal{S}^o(\mathbf{y}_1 \cdots \mathbf{y}_{i-1}) \oplus \mathbf{y}_i^e \oplus \mathbf{y}_i^o - (\mathbf{y}_i^e \oplus \mathcal{S}^o(\mathbf{y}_1 \cdots \mathbf{y}_{i-1}) \oplus \mathbf{y}_i^o)$$

Combining the above two derivations, we have

$$\vdash \mathcal{S}(\mathbf{y}_1 \cdots \mathbf{y}_{i-1}) \oplus \mathbf{y}_i - \mathcal{S}^e(\mathbf{y}_1 \cdots \mathbf{y}_{i-1}) \oplus \mathbf{y}_i^e \oplus \mathcal{S}^o(\mathbf{y}_1 \cdots \mathbf{y}_{i-1}) \oplus \mathbf{y}_i^o$$

Now, using the previous lemma, we are done.

□

The following corollary easily follows from repeated application of the above lemma.

Corollary 9. *For $j < i$,*

$$\vdash \mathcal{S}(\mathbf{y}_1 \cdots \mathbf{y}_i) - \mathcal{S}(\mathbf{y}_1 \cdots \mathbf{y}_j) \oplus \mathcal{S}(\mathbf{y}_{j+1} \cdots \mathbf{y}_i)$$

.

Lemma 28. *For every t*

$$\vdash \mathcal{S}(\mathbf{y}_1 X_1 \cdots \mathbf{y}_t X_t) \oplus \mathcal{S}(\mathbf{z}_1 X_1 \cdots \mathbf{z}_t X_t) \\ - \mathcal{S}((\mathbf{y}_1 \oplus \mathbf{z}_1) X_1 \cdots (\mathbf{y}_t \oplus \mathbf{z}_t) X_t)$$

Proof. Assume by induction that we have made the above derivation until $t = i - 1$. Then we have

$$\begin{aligned}
&\vdash_1 \mathcal{S}(\mathbf{y}_1 X_1 \cdots \mathbf{y}_i X_i) \oplus \mathcal{S}(\mathbf{z}_1 X_1 \cdots \mathbf{z}_i X_i) \\
&\quad - \mathcal{S}(\mathbf{y}_1 X_1 \cdots \mathbf{y}_{i-1} X_{i-1}) \oplus \mathbf{y}_i X_i \oplus \mathcal{S}(\mathbf{z}_1 X_1 \cdots \mathbf{z}_{i-1} X_{i-1}) \oplus \mathbf{z}_i X_i \\
&\vdash_2 \mathcal{S}(\mathbf{y}_1 X_1 \cdots \mathbf{y}_i X_i) \oplus \mathcal{S}(\mathbf{z}_1 X_1 \cdots \mathbf{z}_i X_i) \\
&\quad - \mathcal{S}(\mathbf{y}_1 X_1 \cdots \mathbf{y}_{i-1} X_{i-1}) \oplus \mathcal{S}(\mathbf{z}_1 X_1 \cdots \mathbf{z}_{i-1} X_{i-1}) \oplus \mathbf{y}_i X_i \oplus \mathbf{z}_i X_i \\
&\vdash_3 \mathcal{S}(\mathbf{y}_1 X_1 \cdots \mathbf{y}_i X_i) \oplus \mathcal{S}(\mathbf{z}_1 X_1 \cdots \mathbf{z}_i X_i) \\
&\quad - \mathcal{S}((\mathbf{y}_1 \oplus \mathbf{z}_1) X_1 \cdots (\mathbf{y}_{i-1} \oplus \mathbf{z}_{i-1}) X_{i-1}) \oplus (\mathbf{y}_i \oplus \mathbf{z}_i) X_i \\
&\vdash_4 \mathcal{S}(\mathbf{y}_1 X_1 \cdots \mathbf{y}_i X_i) \oplus \mathcal{S}(\mathbf{z}_1 X_1 \cdots \mathbf{z}_i X_i) \\
&\quad - \mathcal{S}((\mathbf{y}_1 \oplus \mathbf{z}_1) X_1 \cdots (\mathbf{y}_t \oplus \mathbf{z}_t) X_t)
\end{aligned}$$

where $\vdash_1$ and $\vdash_4$ follow by Lemma 27, $\vdash_2$ follows by Lemma 25 and $\vdash_3$ follows by the induction hypothesis. $\square$

Finally, we show how to derive the representation of the sum of two polynomials.

Lemma 29. *Let P and Q be two polynomials. Then $\mathcal{R}(P+Q) = \mathcal{R}(P) \oplus \mathcal{R}(Q)$.*

Proof. Let $X_1 \cdots X_t$ be monomials that occur in both P and Q, such that $P = a_1X_1 + \cdots + a_tX_t + P_1$ and $Q = b_1X_1 + \cdots + b_tX_t + Q_1$. Then from the definition of $\mathcal{R}$ and Corollary 9 we have

$$\vdash \mathcal{R}(P) - \mathcal{S}(\mathbf{a}_1X_1 \cdots \mathbf{a}_tX_t) \oplus \mathcal{R}(P_1)$$
$$\vdash \mathcal{R}(Q) - \mathcal{S}(\mathbf{b}_1X_1 \cdots \mathbf{b}_tX_t) \oplus \mathcal{R}(Q_1)$$

Using the above, we now have

$$\begin{aligned}
&\vdash \mathcal{R}(P) \oplus \mathcal{R}(Q) \\
&\quad - \mathcal{S}(\mathbf{a}_1X_1 \cdots \mathbf{a}_tX_t) \oplus \mathcal{R}(P_1) \oplus \mathcal{S}(\mathbf{b}_1X_1 \cdots \mathbf{b}_tX_t) \oplus \mathcal{R}(Q_1) \\
&\vdash_1 \mathcal{R}(P) \oplus \mathcal{R}(Q) \\
&\quad - \mathcal{S}(\mathbf{a}_1X_1 \cdots \mathbf{a}_tX_t) \oplus \mathcal{S}(\mathbf{b}_1X_1 \cdots \mathbf{b}_tX_t) \oplus \mathcal{R}(P_1) \oplus \mathcal{R}(Q_1) \\
&\vdash_2 \mathcal{R}(P) \oplus \mathcal{R}(Q) \\
&\quad - \mathcal{S}((\mathbf{a}_1 \oplus \mathbf{b}_1)X_1 \cdots (\mathbf{a}_t \oplus \mathbf{b}_t)X_t) \oplus \mathcal{R}(P_1) \oplus \mathcal{R}(Q_1) \\
&\vdash_3 \mathcal{R}(P) \oplus \mathcal{R}(Q) \\
&\quad - \mathcal{R}(P+Q)
\end{aligned}$$

where $\vdash_1$ is by Lemma 25 and $\vdash_2$ is by the previous lemma. $\vdash_3$ is by Corollary 9 and the definition of $\mathcal{R}$.

$\square$

Lemma 30. *For two vectors $\mathbf{y}$ and $\mathbf{z}$, $-(\mathbf{y} \oplus \mathbf{z}) = (-\mathbf{y}) \oplus (-\mathbf{z})$.*

Proof. Let $\mathbf{w} = \mathbf{y} \oplus \mathbf{z}$ and let $\mathbf{y_1}$, $\mathbf{z_1}$ be vectors obtained by flipping the bits of $\mathbf{y}$, $\mathbf{z}$ respectively. Let $\mathbf{w_1} = \mathbf{y_1} \oplus \mathbf{z_1}$. It is easy to derive for every i,

$$\vdash \mathbf{y}(i) \oplus \mathbf{z}(i) - \mathbf{y_1}(i) \oplus \mathbf{z_1}(i) \tag{48}$$

For $j < \xi$, let $b_j = \Big(\wedge_{i<j} \ (\mathbf{y}(i) \oplus \mathbf{z}(i))\Big) \wedge \neg(\mathbf{y}(j) \oplus \mathbf{z}(j))$ and $b_\xi = \wedge_{i\le\xi}(\mathbf{y}(i) \oplus \mathbf{z}(i))$ be a boolean variable indicating the least index i_0 such that $\mathbf{y}(i_0) \oplus \mathbf{z}(i_0) = 0$. Let c_i be the carry bits in $\mathbf{y} \oplus \mathbf{z}$. We translate boolean formulas into polynomials using the operator $tr()$ defined in Section 5.2.1. We first derive for every j and $i \le j$,

$$\vdash tr(b_j \to (c_i = 0))$$

This is done by noting that $c_1 = 0$ and by (45) of Lemma 24, $c_i = H(\mathbf{y}(i-1), \mathbf{z}(i-1), c_{i-1})$ for $i > 1$. Assuming by induction that we have derived for some $j > i \geq 1$

$$\vdash tr(b_j \to (c_i = 0))$$

it is easy to derive

$$\vdash tr(b_j \to \mathbf{y}(i) \oplus \mathbf{z}(i))$$

Now using the above two derivations with the identity (42) of Lemma 24 and the observation $\vdash \mathbf{y}(i) \oplus \mathbf{z}(i) \to \neg H(\mathbf{y}(i), \mathbf{z}(i))$, we have

$$\vdash tr(c_{i+1} - H(c_i, \mathbf{y}(i) \oplus \mathbf{z}(i)) \oplus H(\mathbf{y}(i), \mathbf{z}(i)))$$

$$\vdash tr(b_j \to (c_{i+1} = 0))$$

Since $\mathbf{w}(i) = \mathbf{y}(i) \oplus \mathbf{z}(i) \oplus c_i$, for every j and $i \leq j$, we have the derivation

$$\vdash tr(b_j \to (\mathbf{w}(i) = \mathbf{w_1}(i)))$$

We now want to inductively derive for every j and $i > j$

$$\vdash tr(b_j \to (\mathbf{w}(i) = \mathbf{w}_1(i) \oplus 1)) \tag{49}$$

Let c'_i indicate the carry bits in $\mathbf{y}_1 \oplus \mathbf{z}_1$. Due to the derivation (48), we only need to derive for every $i > j$

$$\vdash tr(b_j \to c_i \oplus c'_i)$$

If $\mathbf{y}(i-1) \oplus \mathbf{z}(i-1) = 0$ (this includes the base case of $i = j+1$), it is easy to derive the following identity independent of the values of c_{i-1} and c'_{i-1}.

$$\vdash tr((\mathbf{y}(i-1) \oplus \mathbf{z}(i-1) = 0) \to c_i \oplus c'_i)$$

Assuming now that we have derived $\vdash c_i \oplus c'_i$ for some $i > j$, for the case where $\mathbf{y}(i-1) \oplus \mathbf{z}(i-1) = 1$, it is easy to derive

$$\vdash tr((c_i \oplus c'_i) \wedge (\mathbf{y}(i) \oplus \mathbf{z}(i) = 0) \to c_{i+1} \oplus c'_{i+1})$$

Now consider the vector $\mathbf{w_1} \oplus \mathbf{1}$. By the definition of b_j we have the derivation for all $i < j$

$$\vdash tr(b_j \to (\mathbf{w}(i) = 1))$$

and

$$\vdash tr(b_j \to (\mathbf{w}(j) = 0))$$

Thus it is easy to derive for $i \leq j$

$$\vdash tr(b_j \to (\mathbf{w} \oplus \mathbf{1}(i) = \mathbf{w}(i) \oplus 1))$$

and for $i > j$

$$\vdash tr(b_j \to (\mathbf{w} \oplus \mathbf{1}(i) = \mathbf{w}(i)))$$

Combining the above two derivations with (49), we have for all i and j

$$\vdash tr(b_j \to (\mathbf{w} \oplus \mathbf{1}(i) = \mathbf{w}(i) \oplus 1))$$

Since b_j are mutually exclusive, we can eliminate them using techniques similar to Lemma 6 and obtain

$$\vdash \mathbf{w} \oplus \mathbf{1}(i) - \mathbf{w}(i) \oplus 1$$

Hence $\mathbf{w_1} \oplus \mathbf{1}$ the vector obtained by flipping all the bits of $\mathbf{w}$. Therefore, using the definition of $-\mathbf{w}$ and Lemma 25

$$\vdash (-\mathbf{w}) - \ \mathbf{y_1} \oplus \mathbf{z_1} \oplus \mathbf{1} \oplus \mathbf{1}$$
$$\vdash (-\mathbf{w}) - (-\mathbf{y}) \oplus (-\mathbf{z})$$

□

Lemma 31. *For any vector* $\mathbf{y}$ *of length* $\ell < \xi - 1$,

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

Proof. Since $\mathbf{y}$ is of length ℓ, we have for $\ell < j \leq \xi$

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

Let $\mathbf{y}_1$ be the vector obtained by flipping the bits of $\mathbf{y}$. Then we have the derivation for $\ell < j \leq \xi$

$$\mathbf{y}(\xi) - 1 \vdash \mathbf{y}_1(j)$$

Now, using the identity (45) of Lemma 24, we have for $\ell + 1 < j \leq \xi$

$$\mathbf{y}(\xi) - 1 \vdash (\mathbf{y}_1 \oplus \mathbf{1})(j)$$

Since $-\mathbf{y} = \mathbf{y}_1 \oplus \mathbf{1}$, the lemma follows.

□

Lemma 32. *Let* P *be a polynomial represented by a vector* $\mathbf{y}$. *Then* $\vdash \mathcal{R}(-P) - (-\mathbf{y})$.

Proof. Let $P = a_1X_1 + \cdots + a_tX_t$. We derive the above by induction on t. Let $P_i = a_1X_1 + \cdots + a_tX_i$ for $i < t$. Then since by Lemma 27, $\vdash \mathcal{R}(P) - (\mathcal{R}(P_{t-1}) \oplus \mathbf{a}_tX_t)$, we have by Lemma 30

$$\vdash (-\mathcal{R}(P)) - (-\mathcal{R}(P_{t-1})) \oplus (-\mathbf{a}_tX_t)$$

The lemma now follows from the induction hypothesis and Lemma 27.

□

D.2 Non-negative vectors are closed under addition

In this section we show that non-negative vectors of bounded length are closed under the addition $\oplus$. This will be used to show that the vector representations of all the lines of the simulation are bounded in length. Note that some of these claims need not be provable in our proof system.

We first show that given two vectors $\mathbf{y}$ and $\mathbf{z}$ of length ℓ, $\mathbf{y} \oplus \mathbf{z}$ is of length at most $\ell + 1$.

Lemma 33. *Given two vectors* $\mathbf{y}$ *and* $\mathbf{z}$ *of length at most* ℓ*,* $\mathbf{w} = \mathbf{y} \oplus \mathbf{z}$ *is of length at most* $\ell + 1$

Proof. Let d_i be the carry to the i^{th} position in $\mathbf{y} \oplus \mathbf{z}$. We branch on the value of $d_{\ell+1}$. If $d_{\ell+1} = 0$, then all the bits at positions greater than ℓ in $\mathbf{w}$ are equal to $s_1 \oplus s_2$ and thus the length of $\mathbf{w}$ is at most ℓ. If $d_{\ell+1} = 1$, then if $s_1 \vee s_2 = 0$, $\mathbf{w}(\ell + 1) = 1$ and $\mathbf{w}(j) = 0$ for $j > \ell + 1$. Thus the length of $\mathbf{w}$ is at most $\ell + 1$. If $s_1 \vee s_2 = 1$, then it is easy to see that $d_j = 1$ and thus $\mathbf{w}(j) = s_1 \oplus s_2 \oplus 1$ for $j \geq \ell + 1$ and thus the length of $\mathbf{w}$ is at most ℓ. □

Lemma 34. *Let* $\mathbf{y}_1 \cdots \mathbf{y}_k$ *be vectors of length* ℓ *such that* $\lceil \log k \rceil + \ell < \xi - 1$*. Then* $\mathcal{S}(\mathbf{y}_1 \cdots \mathbf{y}_k)$ *is of length at most* $\lceil \log k \rceil + \ell$*.*

Proof. Assume that the statement is true for up to $k/2$ vectors. Then by Corollary 9,

$$\vdash \mathcal{S}(\mathbf{y}_1 \cdots \mathbf{y}_k) - \mathcal{S}(\mathbf{y}_1 \cdots \mathbf{y}_{k/2}) \oplus \mathcal{S}(\mathbf{y}_{k/2+1} \cdots \mathbf{y}_k)$$

Now by the induction hypothesis, $\mathcal{S}(\mathbf{y}_1 \cdots \mathbf{y}_{k/2})$ and $\mathcal{S}(\mathbf{y}_{k/2+1} \cdots \mathbf{y}_k)$ are of length at most $\lceil \log k \rceil - 1 + \ell$. Using the previous lemma, we are done. □

Using the observation that for a constant a_1 with bit complexity ℓ, $\mathbf{a}_1X_1$ is a vector of length ℓ, we have the following corollary.

Corollary 10. *Let* $P = a_1X_1 + \cdots + a_tX_t$ *be a polynomial with coefficients of bit length at most* ℓ*. Then* $\mathcal{R}(P)$ *is a vector of length at most* $\ell + \lceil \log t \rceil$

Lemma 35. *For any two vectors* $\mathbf{a}$ *and* $\mathbf{b}$ *of length at most* $\ell < \xi - 1$

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

Proof. Since $\mathbf{a}$ and $\mathbf{b}$ are of length at most ℓ we have for $\xi \geq j > \ell$

$$\mathbf{a}(\xi) \vdash \mathbf{a}(j)$$

$$\mathbf{b}(\xi) \vdash \mathbf{b}(j)$$

Thus there is no carry beyond position $\ell + 1 < \xi$ in $\mathbf{a} \oplus \mathbf{b}$ due to our assumptions and thus using identity (45) of Lemma 24, it is easy to derive

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

□

Since by Lemma 34, the vectors $\mathcal{R}(P_1)$ and $\mathcal{R}(P_2)$ are of length at most $\ell = \lceil \log \xi_0 \rceil + \lceil \log \xi_1 \rceil < \xi - 1$ and by Lemma 29, $\vdash \mathcal{R}(P_1 + P_2) - \mathcal{R}(P_1) \oplus \mathcal{R}(P_2)$, we have the following corollary.

Corollary 11. *For any two polynomials P_1 and P_2 with at most ξ_0 monomials and coefficients of magnitude at most ξ_1,*

$$\mathcal{R}(P_1)(\xi), \mathcal{R}(P_2)(\xi) \vdash \mathcal{R}(P_1 + P_2)(\xi)$$

The following corollary now follows easily from Lemma 27 and the previous lemma.

Corollary 12. *Let $\mathbf{y}_1 \cdots \mathbf{y}_k$ be non-negative vectors of length ℓ such that $\lceil \log k \rceil + \ell < \xi - 1$. Then*

$$\mathbf{y}_1(\xi), \cdots, \mathbf{y}_k(\xi) \vdash \mathcal{S}(\mathbf{y}_1 \cdots \mathbf{y}_k)(\xi)$$

Lemma 36. *Let $\mathbf{y}$ and $\mathbf{z}$ be two non-negative vectors of length ℓ such that $3\ell < \xi - 1$. Then*

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

Proof. Since $\mathbf{z}$ is non-negative of length ℓ, for $\ell + 1 \leq i \leq \xi$

$$\mathbf{z}(\xi) \vdash \mathbf{z}(i)$$

Therefore,

$$\mathcal{SS}(\mathbf{y}, \mathbf{z}) = \mathcal{S}(\mathbf{z}(0)\mathbf{y} \cdots \mathbf{z}(\xi - 1)2^{\xi-1}\mathbf{y}) = \mathcal{S}(\mathbf{z}(0)\mathbf{y} \cdots \mathbf{z}(\ell)2^{\ell}\mathbf{y})$$

Since each of the vectors $\mathbf{z}(0)\mathbf{y}, \cdots\ \mathbf{z}(\ell)2^{\ell}\mathbf{y}$ is of length at most 2ℓ and there are ℓ of them, by the previous corollary, we are done.

□

D.3 Properties of multiplication

Here we show that multiplication is distributive and can be treated as repeated addition.

Lemma 37. *Distributivity of $\mathcal{R}$*
Let P, P_1, P_2, Q be polynomials such that $P = P_1 + P_2$. Then

$$\mathcal{R}(PQ) = \mathcal{R}(P_1Q) \oplus \mathcal{R}(P_2Q)$$

Proof. Easily follows from Corollary 9 □

The following lemmas show that multiplication is repeated addition.

Lemma 38. *Let y, z be two bits and let $\mathbf{w}$ be a vector. Then,*

$$\vdash y\mathbf{w} \oplus z\mathbf{w} - (y \oplus z)\mathbf{w} \oplus H(y, z)2\mathbf{w}$$

Proof. Let $\mathbf{w}_1 = (y \oplus z)\mathbf{w}$ and $\mathbf{w}_2 = H(y, z)2\mathbf{w}$. Let e_i be the carry bit to the i^{th} position in $\mathbf{w}_1 \oplus \mathbf{w}_2$ and let c_i be the carry bit to the i^{th} position in $y\mathbf{w} \oplus z\mathbf{w}$. We will derive by induction that for every i,

$$\vdash (y\mathbf{w} \oplus z\mathbf{w})(i) - (\mathbf{w}_1 \oplus \mathbf{w}_2)(i)$$
$$\vdash e_{i+1} - H(c_i, y\mathbf{w}(i) \oplus z\mathbf{w}(i))$$

This is easy to derive for the case of $i = 1$ since $\mathbf{w}_2(1) = 0$ and thus the first bit on both sides is equal to $(y \oplus z)\mathbf{w}(1)$. Also by (45) of Lemma 24, $e_2 = 0$ is derived since $\mathbf{w}_2(1) = 0$ and therefore there is no carry to the second position. Since $c_1 = 0$, $H(c_1, y\mathbf{w}(1) \oplus z\mathbf{w}(1)) = e_2 = 0$. Now assume that we have derived it up to $i - 1$ for some $i > 1$. Then we have

$$\vdash e_i - H(c_{i-1}, y\mathbf{w}(i-1) \oplus z\mathbf{w}(i-1))$$

and from the definition of $\mathbf{w_2}$ it is easy to derive

$$\vdash \mathbf{w}_2(i) - H(y\mathbf{w}(i-1), z\mathbf{w}(i-1))$$

Therefore by using Identities (42) and (45)

$$\vdash e_1 \oplus \mathbf{w}_2(i) - H(c_{i-1}, y\mathbf{w}(i-1), z\mathbf{w}(i-1)) \tag{50}$$
$$\vdash e_1 \oplus \mathbf{w}_2(i) - c_i$$

And by Identity (44)

$$\vdash H(e_1, \mathbf{w}_2(i)) \tag{51}$$

From the above derivations, we now have

$$\vdash e_i \oplus \mathbf{w}_1(i) \oplus \mathbf{w}_2(i) - y\mathbf{w}(i) \oplus z\mathbf{w}(i) \oplus c_i$$

which derives that the i^{th} bits on both sides are equal.
Also, we have by (45) of Lemma 24

$$\vdash e_{i+1} - H(e_i, \mathbf{w}_1(i), \mathbf{w}_2(i))$$

By identity (42) we have

$$\vdash e_{i+1} - H(e_i, \mathbf{w}_2(i)) \oplus H(\mathbf{w}_1(i), e_i \oplus \mathbf{w}_2(i))$$

and by (50) and (51)

$$\vdash e_{i+1} - H(y\mathbf{w}(i) \oplus z\mathbf{w}(i), c_i)$$

which continues the induction.

□

Lemma 39. *Let $\mathbf{y} = [y_{k-1} \cdots y_0]$ and $\mathbf{z} = [z_{k-1} \cdots z_0]$ be two bit vectors of dimension k, let $\mathbf{w} = \mathbf{y} \oplus \mathbf{z}$ and let d_1 be a constant and X_1 be a monomial. Then,*

$$\vdash \mathcal{SS}(\mathbf{d}_1X_1, \mathbf{w}) - \mathcal{SS}(\mathbf{d}_1X_1, \mathbf{y}) \oplus \mathcal{SS}(\mathbf{d}_1X_1, \mathbf{z})$$

Proof. For the base case where $\mathbf{y}$ and $\mathbf{z}$ are of dimension one, the above derivation follows easily from the previous lemma. Assume that the statement is derived when $\mathbf{y}$ and $\mathbf{z}$ are vectors of dimension $k-1$. Let $\mathbf{y}_{k-1}$, $\mathbf{z}_{k-1}$, $\mathbf{w}_{k-1}$ denote the corresponding vectors truncated to dimension $k-1$ by dropping the element(s) with the highest index. Let e_i be the carry to the i^{th} position in $\mathbf{y} \oplus \mathbf{z}$, i.e. $\mathbf{w}(i) = y_{i-1} \oplus z_{i-1} \oplus e_i$.

By the definition of $\mathcal{SS}(.)$ and Lemma 27, we derive

$$\begin{aligned}\vdash\ & \mathcal{SS}(\mathbf{d}_1X_1, \mathbf{y}) \oplus \mathcal{SS}(\mathbf{d}_1X_1, \mathbf{z}) \\ & - \mathcal{SS}(\mathbf{d}_1X_1, \mathbf{y}_{k-1}) \oplus y_{k-1}2^{k-1}\mathbf{d}_1X_1 \oplus \mathcal{SS}(\mathbf{d}_1X_1, \mathbf{z}_{k-1}) \oplus z_{k-1}2^{k-1}\mathbf{d}_1X_1\end{aligned}$$

By using associativity (Lemma 25), we have

$$\begin{aligned}\vdash\ & \mathcal{SS}(\mathbf{d}_1X_1, \mathbf{y}) \oplus \mathcal{SS}(\mathbf{d}_1X_1, \mathbf{z}) \\ & - \mathcal{SS}(\mathbf{d}_1X_1, \mathbf{y}_{k-1}) \oplus \mathcal{SS}(\mathbf{d}_1X_1, \mathbf{z}_{k-1}) \oplus y_{k-1}2^{k-1}\mathbf{d}_1X_1 \oplus z_{k-1}2^{k-1}\mathbf{d}_1X_1\end{aligned}$$

Now using the previous lemma and the induction hypothesis we derive

$$\begin{aligned}\vdash\ & \mathcal{SS}(\mathbf{d}_1X_1, \mathbf{y}) \oplus \mathcal{SS}(\mathbf{d}_1X_1, \mathbf{z}) \\ & - \mathcal{SS}(\mathbf{d}_1X_1, \mathbf{y}_{k-1} \oplus \mathbf{z}_{k-1}) \oplus (y_{k-1} \oplus z_{k-1})2^{k-1}\mathbf{d}_1X_1 \oplus H(y_{k-1}, z_{k-1})2^{k}\mathbf{d}_1X_1\end{aligned}$$

By the definition of $\mathbf{w}_{k-1}$, it is easy to derive

$$\vdash \mathbf{y}_{k-1} \oplus \mathbf{z}_{k-1} - \mathbf{w}_{k-1} \oplus e_k 2^{k-1}\mathbf{1}$$

Now by Lemma 27 and the definition of $\mathcal{SS}(.)$ we have

$$\begin{aligned}
\vdash\ & \mathcal{SS}(\mathbf{d}_1 X_1, \mathbf{y}) \oplus \mathcal{SS}(\mathbf{d}_1 X_1, \mathbf{z}) \\
& - \Big(\mathcal{SS}(\mathbf{d}_1 X_1, \mathbf{w}_{k-1}) \\
& \oplus e_k 2^{k-1}\mathbf{d}_1 X_1 \\
& \oplus (y_{k-1} \oplus z_{k-1}) 2^{k-1}\mathbf{d}_1 X_1 \\
& \oplus H(y_{k-1}, z_{k-1}) 2^{k}\mathbf{d}_1 X_1\Big)
\end{aligned}$$

By the previous lemma, we can derive

$$\begin{aligned}
\vdash\ & e_k 2^{k-1}\mathbf{d}_1 X_1 \oplus (y_{k-1} \oplus z_{k-1}) 2^{k-1}\mathbf{d}_1 X_1 \\
& - (y_{k-1} \oplus z_{k-1} \oplus e_k) 2^{k-1}\mathbf{d}_1 X_1 \oplus H(y_{k-1} \oplus z_{k-1}, e_k) 2^{k}\mathbf{d}_1 X_1
\end{aligned}$$

Combining this with the above derivation, we have

$$\begin{aligned}
\vdash\ & \mathcal{SS}(\mathbf{d}_1 X_1, \mathbf{y}) \oplus \mathcal{SS}(\mathbf{d}_1 X_1, \mathbf{z}) \\
& - \Big(\mathcal{SS}(\mathbf{d}_1 X_1, \mathbf{w}_{k-1}) \\
& \oplus (y_{k-1} \oplus z_{k-1} \oplus e_k) 2^{k-1}\mathbf{d}_1 X_1 \\
& \oplus H(y_{k-1} \oplus z_{k-1}, e_k) 2^{k}\mathbf{d}_1 X_1 \\
& \oplus H(y_{k-1}, z_{k-1}) 2^{k}\mathbf{d}_1 X_1\Big)
\end{aligned}$$

Now from identities (42) and (44) of Lemma 24

$$\begin{aligned}
\vdash\ & \mathcal{SS}(\mathbf{d}_1 X_1, \mathbf{y}) \oplus \mathcal{SS}(\mathbf{d}_1 X_1, \mathbf{z}) \\
& - \Big(\mathcal{SS}(\mathbf{d}_1 X_1, \mathbf{w}_{k-1}) \\
& \oplus (y_{k-1} \oplus z_{k-1} \oplus e_k) 2^{k-1}\mathbf{d}_1 X_1 \\
& \oplus H(y_{k-1}, z_{k-1}, e_k) 2^{k}\mathbf{d}_1 X_1\Big)
\end{aligned}$$

Noting that $(y_{k-1} \oplus z_{k-1} \oplus e_k)$ and $H(y_{k-1}, z_{k-1}, e_k)$ are equal to $\mathbf{w}(k)$ and $\mathbf{w}(k+1)$ respectively, and using the definition of $\mathcal{SS}(.)$ and Lemma 27 we have

$$\vdash \ \mathcal{SS}(\mathbf{d}_1 X_1, \mathbf{y}) \oplus \mathcal{SS}(\mathbf{d}_1 X_1, \mathbf{z}) - \mathcal{SS}(\mathbf{d}_1 X_1, \mathbf{w})$$

□

Lemma 40. *Let $Q = a'_1 X_1 + \cdots + a'_k X_k$ be represented by a bit vector $\mathbf{z} = [z_{\xi-1} \cdots z_0]$ and let $a_0 X_0$ be a monomial such that the bit length of $a_0 a'_i$ is at most $\xi - 1$. Then*

$$\vdash \mathcal{R}(a_0 X_0 Q) - \mathcal{SS}(a_0 X_0, \mathbf{z})$$

Proof. Let $Q_j = a'_1 X_1 + \cdots + a'_j X_j$ for $j < k$ and let $\mathbf{z}_j = [z^j_{\xi-1} \cdots z^j_0]$ be the equal to $\mathcal{R}(Q_j)$. Assume that we have proved the above statement for Q_j, $j < k$. Then by Lemma 27, $\vdash \mathbf{z} - \mathbf{z}_{k-1} \oplus \mathbf{a}'_k X_k$. Therefore by Lemma 39 we have

$$\vdash \mathcal{SS}(\mathbf{a}_0 X_0, \mathbf{z}) - \mathcal{SS}(\mathbf{a}_0 X_0, \mathbf{z}_{k-1}) \oplus \mathcal{SS}(\mathbf{a}_0 X_0, \mathbf{a}'_k X_k)$$

Since the bit length of $a_0 a'_i$ is at most $\xi - 1$, $\mathcal{SS}(\mathbf{a}_0 X_0, \mathbf{a}'_k X_k) = \mathcal{R}(a_0 a'_k X_0 X_k)$ by definition and by induction,

$$\vdash \mathcal{SS}(\mathbf{a}_0 X_0, \mathbf{z}_{k-1}) - \mathcal{R}(a_0 X_0 Q_{k-1})$$

Therefore we have

$$\vdash \mathcal{SS}(\mathbf{a}_0 X_0, \mathbf{z}) - \mathcal{R}(a_0 X_0 Q_{k-1}) \oplus \mathcal{R}(a_0 a'_k X_0 X_k)$$

which is equal to $\mathcal{R}(a_0 X_0 Q_k)$ by the Distributivity of $\mathcal{R}$.

□

Lemma 41. *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,*

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

Proof. Let $P = a_1 X_1 + \cdots + a_k X_k$, $Q = a'_1 X'_1 + \cdots + a'_k X'_k$ and let P_j be the sum of the first $j < k$ terms of P . Let $\mathbf{y}_i$ denote the bit vector $2^i \mathcal{R}(P)$. Then $\mathcal{SS}(\mathbf{y}, \mathbf{z}) = \mathcal{S}(z_0 \mathbf{y}_0 \cdots z_{\xi-1} \mathbf{y}_{\xi-1})$.

It is easy to derive for vectors $\mathbf{a}$ and $\mathbf{b}$ and any i

$$\vdash 2^i(\mathbf{a} \oplus \mathbf{b}) - 2^i \mathbf{a} \oplus 2^i \mathbf{b}$$

Now by a simple induction using Lemma 27 we derive

$$\vdash 2^i \mathcal{S}(\mathbf{a}_1 X_1 \cdots \mathbf{a}_k X_k) - \mathcal{S}(2^i \mathbf{a}_1 X_1 \cdots 2^i \mathbf{a}_k X_k)$$

$$\vdash \mathbf{y}_i - \mathcal{S}(2^i \mathbf{a}_1 X_1 \cdots 2^i \mathbf{a}_k X_k)$$

Let $\mathbf{y}_i^j = \mathcal{S}(2^i\mathbf{a}_1 X_1 \cdots 2^i\mathbf{a}_j X_j)$ for $j < k$. By Lemma 27, $\mathbf{y}_i = \mathbf{y}_i^{k-1} \oplus 2^i\mathbf{a}_k X_k$. Therefore we have

$$\begin{aligned}\vdash \mathcal{S}(z_0\mathbf{y}_0 \cdots z_{\xi-1}\mathbf{y}_{\xi-1}) \\ - \mathcal{S}(z_0\mathbf{y}_0^{k-1} \oplus z_0\mathbf{a}_k X_k \cdots z_{\xi-1}\mathbf{y}_{\xi-1}^{k-1} \oplus z_{\xi-1}2^{\xi-1}\mathbf{a}_k X_k)\end{aligned}$$

By repeated applications of Lemma 27 we can derive

$$\begin{aligned}\vdash \mathcal{S}(z_0\mathbf{y}_0 \cdots z_{\xi-1}\mathbf{y}_{\xi-1}) \\ - \mathcal{S}(z_0\mathbf{y}_0^{k-1} \cdots z_{\xi-1}\mathbf{y}_{\xi-1}^{k-1}) \oplus \mathcal{S}(z_0\mathbf{a}_k X_k \cdots z_{\xi-1}2^{\xi-1}\mathbf{a}_k X_k)\end{aligned}$$

By the definition of $\mathcal{SS}(.)$ we have $\mathcal{SS}(\mathbf{a}_k X_k, \mathbf{z}) = \mathcal{S}(z_0\mathbf{a}_k X_k \cdots z_{\xi-1}2^{\xi-1}\mathbf{a}_k X_k)$ and by Lemma 40 we have

$$\vdash \mathcal{SS}(\mathbf{a}_k X_k, \mathbf{z}) - \mathcal{R}(\mathbf{a}_k X_k Q)$$

and by induction on k we have

$$\vdash \mathcal{SS}(\mathbf{y}_0^{k-1}, \mathbf{z}) - \mathcal{R}(P_{k-1}Q)$$

Thus we derive

$$\vdash \mathcal{S}(z_0\mathbf{y}_0 \cdots z_{\xi-1}\mathbf{y}_{\xi-1}) - \mathcal{R}(P_{k-1}Q) \oplus \mathcal{R}(\mathbf{a}_k X_k Q)$$

The lemma now follows from Distributivity of $\mathcal{R}$. □

References

[1] Yaroslav Alekseev, Dima Grigoriev, Edward A Hirsch, and Iddo Tzameret. 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. A note on the power of threshold circuits. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 580–584. IEEE, 1989.

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

[4] Maria Luisa Bonet, Toniann Pitassi, and Ran Raz. On interpolation and automatization for frege systems. *SIAM Journal on Computing*, 29(6):1939–1967, 2000.

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

[6] Samuel Buss, Leszek Kołodziejczyk, and Konrad Zdanowski. Collapsing modular counting in bounded arithmetic and constant depth propositional proofs. *Transactions of the American Mathematical Society*, 367(11):7517–7563, 2015.

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

[8] Samuel R. Buss, Russell Impagliazzo, Jan Krajícek, Pavel Pudlák, Alexander A. Razborov, and Jirí Sgall. Proof complexity in algebraic systems and bounded depth frege systems with modular counting. *Computational Complexity*, 6(3):256–298, 1997.

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

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

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

[12] Mikael Goldmann, Johan Håstad, and Alexander A. Razborov. Majority gates VS. general weighted threshold gates. *Computational Complexity*, 2:277–300, 1992.

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

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

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

[16] Joshua A. Grochow and Toniann Pitassi. Circuit complexity, proof complexity, and polynomial identity testing: The ideal proof system. *J. ACM*, 65(6):37:1–37:59, 2018.

[17] Pavel Hrubes and Iddo Tzameret. 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*, pages 41–51, 2009.

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

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

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

[21] Alexis Maciel and Denis Thérien. Threshold circuits of small majority-depth. *Inf. Comput.*, 146(1):55–83, 1998.

[22] Toniann Pitassi. Algebraic propositional proof systems. In *Descriptive Complexity and Finite Models, Proceedings of a DIMACS Workshop 1996, Princeton, New Jersey, USA, January 14-17, 1996*, pages 215–244, 1996.

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

[24] Toniann Pitassi and Rahul Santhanam. Effectively polynomial simulations. In *ICS*, pages 370–382, 2010.

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

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

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

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

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

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il