Uniform, Integral and Efficient Proofs for the Determinant Identities
(Extended Abstract)

Iddo Tzameret
Department of Computer Science
Royal Holloway, University of London
Iddo.Tzameret@rhul.ac.uk

Stephen A. Cook
Department of Computer Science
University of Toronto
sacook@cs.toronto.edu

***Abstract*—We give a uniform and integral version of the short propositional proofs for the determinant identities demonstrated over $GF(2)$ in Hrubeš-Tzameret [9]. Specifically, we show that the multiplicativity of the determinant function over the integers is provable in the bounded arithmetic theory $\mathbf{VNC}^2$, which is a first-order theory corresponding to the complexity class $\mathbf{NC^2}$. This also establishes the existence of uniform polynomial-size and $O(\log^2 n)$-depth Circuit-Frege (equivalently, Extended Frege) proofs over the integers, of the basic determinant identities (previous proofs hold only over $GF(2)$).**

In doing so, we give *uniform* $\mathbf{NC^2}$*-algorithms* for homogenizing algebraic circuits, balancing algebraic circuits (given as input an upper bound on the syntactic-degree of the circuit), and converting circuits with divisions into circuits with a single division gate—all (Σ_1^B-) definable in $\mathbf{VNC}^2$. This also implies an $\mathbf{NC^2}$-algorithm for *evaluating* algebraic circuits of any depth.[1]

I. Introduction

This work fills a basic gap in our understanding of weak formal theories of arithmetic, namely, bounded arithmetic. The complexity of linear algebraic operations such as matrix inverse and the determinant is well studied [5]. The importance of linear algebra in bounded arithmetic and proof complexity has also been identified in many works, and it has been conjectured for quite a long time (cf. [14], [15], [4], [6], as well as [2], [1]) that the determinant identities, and specifically the multiplicativity of the determinant function $\mathrm{DET}(A)\cdot\mathrm{DET}(B) = \mathrm{DET}(AB)$, for two matrices A, B, can be proved in a formal theory that, loosely speaking, reasons with $\mathbf{NC^2}$ concepts. Here, $\mathbf{NC^2}$ is the complexity class consisting of all languages that can be decided by families of $O(\log^2 n)$-depth and polynomial-size circuits, and is apparently the smallest circuit class *in the* **NC** *hierarchy* that can *compute* the determinant.[2] This conjecture is aligned with the intuition that basic properties of many constructions and functions of a given complexity class can be proved in logical theories using only concepts from the same class.

[1]Our $\mathbf{NC^2}$ evaluation algorithm is different from the previously known one by Miller *et al*. [12]. Our algorithm also requires as input an upper bound on the syntactic-degree of the circuit, while Miller *et al*.'s does not.

[2]Formally, the conjectural smallest circuit class computing integer determinants is the class denoted DET (cf. [6]).

978-1-5090-3018-7/17/$31.00 ©2017 IEEE

Currently, the weakest theory known to prove the determinant identities is $\mathbf{PV}$ (which corresponds to polynomial time reasoning, by Soltys and Cook [15]). Quite recently, Hrubeš and Tzameret [9] showed that at least in the *propositional* case, the determinant identities expressing the multiplicativity of the determinant over $GF(2)$ can be proved with polynomial-size propositional proofs operating with $\mathbf{NC^2}$-circuits (and quasipolynomial size Frege proofs). However, this does not lend itself immediately to the uniform framework of bounded arithmetic. For example, a short propositional proof may be shown to exist, but with no way of determining whether it could be constructed uniformly and in a restricted computational model such as uniform-$\mathbf{NC^2}$ algorithms—making it thus impossible to carry out directly in bounded arithmetic. Further, [9] crucially used in their construction elimination of division gates from algebraic circuits, which we do not know how to achieve using uniform weak computational models like uniform-$\mathbf{NC^2}$ (since for standard division elimination one needs to find field assignments that do not nullify a given polynomial [16]).

The main goal of this work is to prove the determinant identities in the theory $\mathbf{VNC}^2$ (corresponding to "$\mathbf{NC^2}$-reasoning"; i.e., reasoning with concepts definable in the complexity class $\mathbf{NC^2}$). We will show that similar reasoning as in [9] can be carried over, with further complications imposed by uniformity and parallelism, to $\mathbf{VNC}^2$. As a result of working in bounded arithmetic it will also become relatively simpler to conclude the proofs over the integers (while the previous propositional proofs considered only $GF(2)$).

a) Organization: The preliminaries for this work are quite long. *For this reason we begin with a* high-level *overview of the results and their proofs in Sec. II* (readers who are unfamiliar with some of the concepts in the overview can consult the preliminaries section for those). The preliminaries themselves are given in Sec. III, consisting of basic definitions from bounded arithmetic, the complexity class $\mathbf{NC^2}$, the corresponding theory $\mathbf{VNC}^2$ [4], basic definitions of algebraic circuits, and proof systems operating with algebraic circuits establishing polynomial identities (PI-proofs [8], [9]). In Sec. IV we give a much more detailed guide to the proof of the determinant identities in the theory, while still leaving out many of the technical details and proofs. In Sec. V we discuss some of the parallel (uniform-$\mathbf{NC^2}$) algorithms we construct: division-gates normal forms (moving division gates to the root of circuits) and balancing of algebraic circuits. As a result of these algorithms we obtain a parallel algorithm for evaluating algebraic circuits (of any depth). Sec. VI explains in some detail how we encode certain algebraic circuits in the theory.

II. Overview

Our aim is to prove the determinant identities inside $\mathbf{VNC}^2$ (for $\mathbf{VNC}^2$ and the logical setting see Sec. III). Specifically, we want to have a $\mathbf{\Sigma}^B_1$-definable in $\mathbf{VNC}^2$ function $\mathrm{DET}(\cdot)$ whose input is a matrix over the integers, such that $\mathbf{VNC}^2$ proves:

$$\mathrm{DET}(A) \cdot \mathrm{DET}(B) = \mathrm{DET}(AB) \tag{1}$$

and

$$\mathrm{DET}(C) = c_{11} \cdots c_{nn}, \tag{2}$$

for any $n \times n$ integer matrices A, B, and any C an $n \times n$ triangular integer matrix.

Note that these two identities can be considered as the *defining* identities of the determinant polynomial, in the sense that every polynomial for which these two identities hold is the determinant polynomial. (One way of seeing this is to observe that every square matrix is equal to a product of upper and lower triangular matrices.)

Some clarification on how we represent integers and matrices in the theory is due: integer numbers are presented in binary as *strings*, where the least significant bit (lsb) is 0 (resp. 1) when the integer is positive (resp. negative), and where the rest of the string is the binary representation of the absolute value of the integer. An $n \times n$ matrix over $\mathbb{Z}$ is usually encoded as a two-dimensional string (cf. [6]).

It is not hard to show that we can prove *simple* facts about matrices, such as the definability of matrix product AB, the statement expressing associativity and commutativity of matrix products $A(BC) = (AB)C$ and $A + B = B + A$, resp., and so forth (see, e.g., [15], [4] and Lemma 28 in [9] about these basic identities that can be proved already in $\mathbf{VNC}^1$).

All circuit classes discussed in this work (except when otherwise stated) are assumed to be uniform circuit classes (formally, we require uniformity in the sense that the extended connection language of the circuit family is in $\mathbf{FO}$; see [4] and Sec. III-B).

Let us now sketch briefly how we *define* the determinant function in the theory and how we *prove* its defining identities in the theory.

b) Defining the determinant function in the theory: Given a matrix over the integers of dimension $n \times n$, the $\mathbf{\Sigma}^B_1$-definable string function (recall that we encode integers as strings) in $\mathbf{VNC}^2$ for the determinant is defined roughly as follows: first, construct an $O(\log^2 n)$-depth algebraic circuit computing the determinant of $n \times n$ integer matrices, and then evaluate the circuit under the input assignment.

More specifically, the determinant function in the theory first constructs a recursive algebraic circuit (or equivalently, a straight-line program) computing the symbolic $n \times n$ determinant *with division gates* ("symbolic" here means that the algebraic circuit computes the determinant as the formal polynomial over n^2 distinct variables). This is done using the standard recursive formula of the block-wise determinant (using "Schur complement"), simulating in a sense Gaussian elimination (similar to [9]). Then, *eliminate the division gates* in the determinant circuit using, among other conversions, substitutions of power series in the circuit. Then, *homogenize* the circuit getting rid of high degrees, *balance the circuit* to achieve the squared logarithmic depth, and finally *evaluate* the result under the input integer matrix.

The function that evaluates a balanced algebraic circuit in itself consists of several steps, as follows: given as an input a balanced algebraic circuit, the function: (i) converts it into a layered circuit (namely, a circuit in which each node connects only to the subsequent layer); (ii) transforms it into a *Boolean circuit* computing the same polynomial over the integers (coded as bit-strings) while taking care that the negations appear only in the bottom layer; and finally (iii) evaluates the Boolean circuit using the fact that the $\mathbf{NC^2}$-Circuit Evaluation Problem is $\mathbf{NC^2}$-complete (under $\mathbf{AC}^0$ reductions [4]).

Since we show that the determinant function as defined above is $\mathbf{\Sigma}^B_1$-definable in $\mathbf{VNC}^2$, by [4] it means that this function is in uniform-$\mathbf{NC^2}$.

c) Proving the determinant equalities in the theory: Informally, the basic argument formalized in the theory is that there exists a balanced PI-proof (for *Polynomial Identity proof*), in symbols, a $\mathbb{P}_c(\mathbb{Z})$-proof (as in [9]; see Section III-E), of these identities. Thus, by soundness of balanced $\mathbb{P}_c(\mathbb{Z})$-proofs, which we show is provable in $\mathbf{VNC}^2$, these identities must be true.

More precisely, we demonstrate a $\mathbf{\Sigma}^B_1$-definable function in $\mathbf{VNC}^2$ that given an input n in unary, outputs a $\mathbb{P}_c(\mathbb{Z})$-proof of identities (1) and (2). In this $\mathbb{P}_c(\mathbb{Z})$-proof every proof-line is an equation between depth-$O(\log^2(n))$ algebraic circuits (without division gates) of a polynomial syntactic-degree. To conclude the argument, we use the soundness of depth-$O(\log^2(n))$ $\mathbb{P}_c(\mathbb{Z})$-proofs: using induction on proof-length we argue that for *every assignment of integers*, equations (1) and (2) must hold.

A. Contributions and Technical Challenges

Showing that the long and nontrivial constructions from [9] can be carried out in $\mathbf{VNC}^2$ and thus proving the determinant identities, requires quite a lot of work. The main technical obstacles that we face are *parallelism* and *uniformity* as we explain in what follows.

Parallelism here means that the construction of the original propositional proofs from [9] *must be done by itself in* $\mathbf{NC^2}$. The construction in [9] is quite involved, and to make it parallel we need to devise several $\mathbf{NC^2}$-algorithms, all $\mathbf{\Sigma}^B_1$-definable in $\mathbf{VNC}^2$, as follows:

(i) Parallel division normalization: converting algebraic circuits with division gates into circuits with a single division gate at the output gate; **(ii)** Converting algebraic circuits C into the sum of their homogeneous components, given as input an upper bound on the syntactic-degree of C; i.e., each summand $C^{(i)}$ is a homogeneous circuit computing the degree i homogeneous component of C; **(iii)** Balancing an algebraic circuit of size s and depth d into a $\mathrm{poly}(s,d)$-size algebraic circuit of depth $O(\log s \cdot \log d + \log^2 d)$, given as input an upper bound on the syntactic-degree of C.

By first balancing an input circuit and then evaluating it (both in $\mathbf{NC^2}$) our results give rise to: **(iv)** an $\mathbf{NC^2}$ evaluation procedure for algebraic circuits of any depth (given as input an upper bound on their syntactic-degree in unary) that is different from the previously known algorithm by Miller *et al.* [12] (their algorithm does not require the syntactic-degree as input).

The algorithm for **(i)** above is somewhat nontrivial, while the algorithm for **(ii)** follows by observing that the standard Strassen [16] homogenization procedure is parallel in nature (assuming one knows the syntactic-degree of the circuit). The algorithm for **(iii)** follows by a combination of the original balancing procedure by Valiant *et al.* [17] with ideas from [12]. For **(iv)** we need again to combine ideas from [12] and other results and observations.

Proving parallel algorithms for structural results on algebraic circuits is however not enough. We moreover need to show that corresponding constructions also apply *on proofs*, in order to conclude that $\mathbf{VNC}^2$ proves the existence of a (uniform $\mathbf{NC^2}$) function that constructs the (low depth PI-) proofs of the determinant identities. This requires more work.

Uniformity here means that we need the whole proof to be constructible in uniform-$\mathbf{NC^2}$. For instance, we need to eliminate division gates from certain algebraic circuits *uniformly*. To eliminate division gates like u/v (for two nodes u, v), one needs to find an assignment to the variables in which the polynomial computed at node v is nonzero. In general we do *not* know how to do this in the theory. Nevertheless, we show that for our purposes it is enough to eliminate only those division gates that occur in some specific circuits. In order to eliminate division gates we will also need to find 'inverse elements' in the ring of integers, and hence we will show that for our purpose it is enough to consider only the inverse of 1 in $\mathbb{Z}$.

Apart from uniformity and parallelism, working in bounded arithmetic allows us to work more easily over the integers, where previously short $\mathbf{NC^2}$-Frege proofs of the determinant identities were known only over $GF(2)$ [9].

III. Preliminaries

In this section we present some of the necessary logical setting from bounded arithmetic and algebraic circuit complexity. Specifically, we describe the two-sorted bounded arithmetic theory $\mathbf{VNC}^2$ as developed by Cook and Nguyen [4] and show how to define the evaluation of arithmetic circuits over the integers in the theory, and then define algebraic circuits computing formal polynomials and proof systems for polynomial identities [8], [9] (cf. [13] for a survey). We start with an exposition of bounded arithmetic.

Bounded arithmetic is a general name for weak systems of arithmetic, namely, fragments of Peano Arithmetic. The bounded arithmetic theories we use are first-order two-sorted theories, having a first-sort for natural numbers and a second-sort for finite sets of numbers, representing bit-strings via their characteristic functions (for the original *single-sort* treatment of theories of bounded arithmetic see also [3], [7], [11]). The theory $\mathbf{V}^0$ corresponds to the complexity class uniform $\mathbf{AC}^0$, and $\mathbf{VNC}^2$ corresponds to uniform-$\mathbf{NC^2}$. The complexity classes $\mathbf{AC}^0$, $\mathbf{NC^2}$, and their corresponding function classes $\mathbf{FAC}^0$ and $\mathbf{FNC}^2$ are also defined using a two-sorted universe (specifically, the first-ordered sort [numbers] are given to the machines in unary representation and the second-sort as binary strings).

Definition 1 (Language of two-sorted arithmetic $\mathcal{L}_A^2$)**.** *The language of two-sorted arithmetic, denoted $\mathcal{L}_A^2$, consists of the following relation, function and constant symbols:*

$$\{+, \cdot, \leq, 0, 1, |\ |, =_1, =_2, \in\}.$$

We describe the intended meaning of the symbols by considering the standard model $\mathbb{N}_2$ of two-sorted Peano Arithmetic. It consists of a first-sort universe $U_1 = \mathbb{N}$ and a second-sort universe U_2 of all finite subsets of $\mathbb{N}$. The constants 0 and 1 are interpreted in $\mathbb{N}_2$ as the appropriate natural numbers zero and one, respectively. The functions $+$ and $\cdot$ are the usual addition and multiplication on the universe of natural numbers, respectively. The relation $\leq$ is the appropriate "less or equal than" relation on the first-sort universe. The function $|\cdot|$ maps a finite set of numbers to its largest element plus one. The relation $=_1$ is interpreted as equality between numbers, $=_2$ is interpreted as equality between finite sets of numbers. The relation $n \in N$ holds for a number n and a finite set of numbers N if and only if n is an element of N.

We denote the first-sort (number) variables by lower-case letters $x, y, z, \ldots$, and the second-sort (string) variables by capital letters $X, Y, Z, \ldots$. We build formulas in the usual way, using two sorts of quantifiers: number quantifiers and string quantifiers. A number quantifier is said to be *bounded* if it is of the form $\exists x(x \leq t \wedge \ldots)$ or $\forall x(x \leq t \rightarrow \ldots)$, respectively, for some *number term* t that does not contain x. We abbreviate $\exists x(x \leq t \wedge \ldots)$ and $\forall x(x \leq t \rightarrow \ldots)$ by $\exists x \leq t$ and $\forall x \leq t$, respectively. A string quantifier is said to be *bounded* if it is of the form $\exists X(|X| \leq t \wedge \ldots)$ or $\forall X(|X| \leq t \rightarrow \ldots)$ for some *number term* t that does not contain X. We abbreviate $\exists X(|X| \leq t \wedge \ldots)$ and $\forall X(|X| \leq t \rightarrow \ldots)$ by $\exists X \leq t$ and $\forall X \leq t$, respectively.

A formula is in the class of formulas Σ_0^B or Π_0^B if it uses *no string quantifiers* and all number quantifiers are bounded. A formula is in Σ_{i+1}^B or Π_{i+1}^B if it is of the form $\exists X_1 \leq t_1 \ldots \exists X_m \leq t_m \psi$ or $\forall X_1 \leq t_1 \ldots \forall X_m \leq t_m \psi$, where $\psi \in \Pi_i^B$ and $\psi \in \Sigma_i^B$, respectively, and t_i does not contain X_i, for all $i = 1, \ldots, m$. We write $\forall\Sigma_0^B$ to denote the universal closure of Σ_0^B (i.e., the class of Σ_0^B-formulas that possibly have [not necessarily bounded] universal quantifiers on their front [left]). We usually write $T(t)$ to abbreviate $t \in T$, for a number term t and a string term T.

As mentioned before, a finite set of natural numbers N represents a finite string $S_N = S_N^0 \dots S_N^{|N|-1}$ such that $S_N^i = 1$ if and only if $i \in N$. We will abuse notation and identify N and S_N.

A. The Theory $\mathbf{V}^0$

The base theory $\mathbf{V}^0$, which corresponds to the computational class $\mathbf{AC}^0$, consists of the following axioms:

Basic 1. $x + 1 \neq 0$ **Basic 2**. $x + 1 = y + 1 \rightarrow x = y$

Basic 3. $x + 0 = x$ **Basic 4**. $x + (y + 1) = (x + y) + 1$

Basic 5. $x \cdot 0 = 0$ **Basic 6**. $x \cdot (y + 1) = (x \cdot y) + x$

Basic 7. $(x \leq y \wedge y \leq x) \rightarrow x = y$ **Basic 8**. $x \leq x + y$

Basic 9. $0 \leq x$ **Basic 10**. $x \leq y \vee y \leq x$

Basic 11. $x \leq y \leftrightarrow x < y + 1$

Basic 12. $x \neq 0 \rightarrow \exists y \leq x (y + 1 = x)$

L1. $X(y) \rightarrow y < |X|$ **L2**. $y + 1 = |X| \rightarrow X(y)$

SE. $(|X| = |Y| \wedge \forall i \leq |X| \, (X(i) \leftrightarrow Y(i))) \rightarrow X = Y$

Σ_0^B**-COMP.** $\exists X \leq y \forall z < y \, (X(z) \leftrightarrow \varphi(z))$, for all $\varphi \in \Sigma_0^B$ where X does not occur freely in φ.

Here, the axioms **Basic 1** through **Basic 12** are the usual axioms used to define Peano Arithmetic without induction (PA$^-$), which settle the basic properties of addition, multiplication, ordering, and of the constants 0 and 1. The Axiom **L1** says that the length of a string coding a finite set is an upper bound to the size of its elements. **L2** says that $|X|$ gives the largest element of X plus 1. **SE** is the extensionality axiom for strings which states that two strings are equal if they code the same sets. Finally, Σ_0^B**-COMP** is the comprehension axiom *scheme* for Σ_0^B-formulas (i.e., it is an axiom for each such formula) and implies the existence of all sets which contain exactly the elements that fulfill any given Σ_0^B property.

Proposition 1 (Corollary V.1.8. [4])**.** *The theory* $\mathbf{V}^0$ *proves the (number) induction axiom scheme for* Σ_0^B*-formulas* Φ*:*

$$(\Phi(0) \wedge \forall x \, (\Phi(x) \rightarrow \Phi(x + 1))) \rightarrow \forall z \, \Phi(z).$$

In the above induction axiom, x is a number variable and Φ can have additional free variables of both sorts.

We seek to define the determinant function in (some) theory via a $\mathbf{\Sigma}_1^B$-formula, where a function is said to be *defined in a theory* if the theory can prove that given an input to the function there exists a unique output.

B. The Complexity Class $\mathbf{NC^2}$

The uniform complexity class $\mathbf{NC^2}$ is defined using an alternating time-space (nondeterministic) Turing machine.

d) Alternating Turing machines: An alternating Turing machine is a *nondeterministic* Turing machine in which every state, except the halting states, is either an *existential state* or a *universal state*. A *computation* in such a machine can be viewed as an (unbounded fan-in) tree of configurations as follows. A configuration is said to be *existential* (resp. *universal*) if its state is existential (resp. universal). In a computation tree of an alternating Turing machine every *existential* configuration has one or more children, such that each child is a configuration reachable in one step from the configuration in the parent node; and every universal configuration has as its set of children *all* configurations reachable in one step from the configuration on the parent in node. We say that a computation of an alternating Turing machine is *accepting* when all the leaves of the computation tree are accepting configurations. We say that an alternating Turing machine *accepts an input* x if there *exists* an accepting computation tree whose root is the initial configuration with the input x.

A computation tree is said to have k *alternations* if the number of alternations between existential and universal states in every branch of the tree is at most k. An alternating Turing machine is said to *work in* $f(n)$ *alternations* if for every input x of length n the number of alternations in *every* computation tree of x is at most $f(n)$. A computation tree is said to have *space* s if the working space used in every configuration of the tree is at most s. An alternating Turing machine is said to *work in space* $g(n)$ if for every input x of length n the space of every computation tree of x is at most $g(n)$.

Definition 2 (Uniform $\mathbf{NC^2}$)**.** *The uniform complexity class* $\mathbf{NC^2}$ *is defined to be the class of languages that can be decided by alternating Turing machines with* $O(\log n)$ *space and* $O(\log^2 n)$ *time.*

We define the function class $\mathbf{FNC}^2$ as the function class containing all number functions $f(\vec{x}, \vec{X})$ and string functions $F(\vec{x}, \vec{X})$, where $\vec{x}$ and $\vec{X}$ are number and string variables, respectively, such that the relation of the function is defined (resp. bit-defined) in $\mathbf{NC^2}$ (a binary relation R is *defined in* $\mathbf{NC^2}$ if the language containing the set of pairs in R is decidable in $\mathbf{NC^2}$).

e) $\mathbf{NC^2}$ *Boolean circuit families:* Let $\{C_n\}_{n=1}^{\infty}$ be a family of Boolean circuits (with fan-in at most two $\vee, \wedge, \neg$ gates). We say that this family is an $\mathbf{NC^2}$ *circuit family* if every circuit C_n in the family has depth $O(\log^2 n)$ and size $n^{O(n)}$. A circuit taken from a given Boolean $\mathbf{NC^2}$ circuit family is said to be an $\mathbf{NC^2}$*-circuit*. It is known that the $\mathbf{NC^2}$ circuit value problem is complete under $\mathbf{AC}^0$-reductions for the class $\mathbf{NC^2}$ (Definition 2). We say that $\{C_n\}_{n=1}^{\infty}$ is a ***uniform*** $\mathbf{NC^2}$***-circuit family*** if its extended connection language is in $\mathbf{FO}$ (we refer the reader to [4, page 455] for the definitions). This definition coincides with Definition 2.

For the definition of uniform $\mathbf{NC^1}$ (and $\mathbf{AC}^1$) we refer the reader to [4].

C. The Theory $\mathbf{VNC}^2$

Here we define the theory $\mathbf{VNC}^2$ as developed in [4]. It is an extension of $\mathbf{AC}^0$ over the language $\mathcal{L}_A^2$ where we add the axiom stating the existence of a sequence of values that represent the evaluation of monotone Boolean circuits of $O(\log^2(n))$ depth. It is known (cf. [4]) that the Monotone Boolean Circuit Value problem for circuits of $O(\log^2(n))$-depth is complete under $\mathbf{AC}^0$ reduction for $\mathbf{NC^2}$.

The $\mathbf{NC^2}$ CIRCUIT VALUE PROBLEM is the problem that determines the value computed by a Boolean $\mathbf{NC^2}$-circuit, given a 0-1 assignment to its input variables. An input circuit to the problem is encoded as a *layered circuit* with $d+1$ layers, namely, a circuit in which every node in layer j is connected only to zero or more nodes in layer $j+1$. The actual evaluation of such an ($\mathbf{NC^2}$) circuit within the class $\mathbf{NC^2}$ is done in stages, where we start from layer 0 and "compute" (using alternations and nondeterminism) the values of every node in every layer. Formally, we define this evaluation process as follows (see also [4, Chap. IX.5.6]).

The layered monotone Boolean circuit with $d+1$ layers is encoded with a string variable I, with $|I| \le n$, which defines the (Boolean) input gates to the circuit. Then we have a string variable G such that $G(x,y)$, for $x \in [d]$, holds iff the yth gate in layer x is $\wedge$, and is $\vee$ otherwise. Also the wires of C are encoded by a three-dimensional array, namely a string variable E such that $E(z,x,y)$ holds iff the output of gate x on layer z is connected to the input of gate y on layer $z+1$. To compute the value of each of the gates in the circuit C on input I, simply compute the values of the gates in each layer, starting from the input layer, in $d+1$ stages, using the values of the previous layer. The formula $\delta_{LMCV}(n,d,E,G,I,Y)$ below formalises this evaluation procedure (where *LMCV* stands for "layered monotone circuit value"). The two-dimensional array Y stores the result of computation: for $1 \le z \le d$, row $Y^{[z]}$ contains the gates on layer z that output 1.

$$\begin{aligned}&\delta_{LMCV}(n,d,E,G,I,Y) \equiv \\ &\qquad \forall x < n \forall z < d\big((Y(0,x) \leftrightarrow I(x)) \wedge \\ &\big(Y(z+1,x) \leftrightarrow \big(G(z+1,x) \wedge \forall u < n, E(z,u,x) \to \\ &Y(z,u)\big) \vee (\neg G(z+1,x) \wedge \exists u < n, E(z,u,x) \wedge Y(z,u))\big)\big). \end{aligned} \tag{3}$$

The following formula states that the circuit with underlying graph (n,d,E) has fan-in two:

$$\begin{aligned}&Fanin2(n,d,E) \equiv \\ &\quad \forall z < d\, \forall x < n \exists u_1 < n \exists u_2 < n \exists v < n(E(z,v,x) \to \\ &\qquad (v = u_1 \vee v = u_2)\big). \end{aligned} \tag{4}$$

Finally, we arrive at the definition of $\mathbf{VNC}^2$:

Definition 3 ($\mathbf{VNC}^2$)**.** *The theory* $\mathbf{VNC}^2$ *has vocabulary* $\mathcal{L}_A^2$ *and is axiomatized by* $\mathbf{V}^0$ *and the axiom:*

$$\begin{aligned}&Fanin2(n,|n|^2,E) \to \\ &\qquad \exists Y \le \langle |n|^2+1, n\rangle \delta_{LMCV}(n,|n|^2,E,G,I,Y). \end{aligned} \tag{5}$$

Theorem 2. ([4, Corollary IX.5.31]) A function is Σ_1^B-definable in $\mathbf{VNC}^2$ iff it is in $\mathbf{FNC}^2$.

D. Polynomials and Algebraic Circuits

Let $\mathbb{G}$ be a ring. Denote by $\mathbb{G}[X]$ the ring of (commutative) polynomials with coefficients from $\mathbb{G}$ and variables $X := \{x_1, x_2, \dots\}$. A *polynomial* is a formal linear combination of monomials, where a *monomial* is a product of variables. Two polynomials are *identical* if all their monomials have the same coefficients. The *degree* of a polynomial is the maximal total degree of a monomial in it.

Algebraic circuits and formulas over the ring $\mathbb{G}$ compute polynomials in $\mathbb{G}[X]$ via addition and multiplication gates, starting from the input variables and constants from the field. More precisely, an *algebraic circuit* C is a finite directed acyclic graph (DAG) with *input nodes* (i.e., nodes of in-degree zero) and a single *output node* (i.e., a node of out-degree zero). Input nodes are labeled with either a variable or a field element in $\mathbb{F}$. All the other nodes have in-degree two (unless otherwise stated) and are labeled by either an addition gate $+$ or a product gate $\times$. An input node is said to *compute* the variable or scalar that labels itself. A $+$ (or $\times$) gate is said to compute the addition (product, resp.) of the (commutative) polynomials computed by its incoming nodes. An algebraic circuit is called a *formula*, if the underlying directed acyclic graph is a tree (that is, every node has at most one outgoing edge). The *size* of a circuit C is the number of nodes in it, denoted $|C|$, and the *depth* of a circuit is the length of the longest directed path in it.

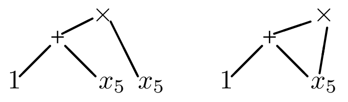
We say that a polynomial is *homogeneous* whenever every monomial in it has the same (total) degree.

Definition 4 (Syntactic-degree $\deg(\cdot)$)**.** *Let C be a circuit and v a node in C. The* syntactic-degree $\deg(v)$ *of v is defined as follows:*

1) *If v is a field element or a variable, then* $\deg(v) := 0$ *and* $\deg(v) := 1$, *respectively;*
2) *If $v = u + w$ then* $\deg(v) := \max\{\deg(u), \deg(w)\}$*;*
3) *If $v = u \cdot w$ then* $\deg(v) := \deg(u) + \deg(w)$.

An algebraic circuit is said to be *syntactic-homogeneous* if for every plus gate $u+v$, $\deg(u) = \deg(v)$.

For an algebraic circuit F we denote by $\widehat{F}$ the polynomial computed by F. We say that two algebraic circuits F, F' are *similar* if F and F' are syntactically identical when both are un-winded into *formulas* (a circuit is un-winded into a formula by duplicating every node in the directed acyclic graph that has a fan-out bigger than one, obtaining a tree instead of a DAG). The similarity relation can be decided in polynomial time (cf. [10]). For example, the following two circuits are similar, since the formula to the left is obtained by "un-winding" the circuit to the right into a formula (cf. [9]):



E. Polynomial Identities (PI) Proofs

In this section we give the necessary background on the PI proof system $\mathbb{P}_c$. This proof was first introduced in [8] (under the name "arithmetic proofs" and for algebraic formulas instead of algebraic circuits), and was subsequently studied in [9].

PI-proofs, as originally introduced in [8], denoted $\mathbb{P}_c$ (and $\mathbb{P}_c(\mathbb{G})$ when we wish to be explicit about the ring $\mathbb{G}$), are sound and complete proof systems for the set of polynomial identities of $\mathbb{G}$, written as equations between algebraic circuits. A PI-proof starts from axioms like associativity, commutativity of addition and product, distributivity of product over addition, unit element axioms, etc., and derives new equations between algebraic circuits $F = G$ using rules for adding and multiplying two previous identities. The axioms of $\mathbb{P}_c$ express reflexivity of equality, commutativity and associativity of addition and product, distributivity, zero element, unit element, and true identities in the field.

Algebraic circuits in PI proofs are treated as purely syntactic objects (similar to the way a propositional formula is a syntactic object in propositional proofs). Thus, simple computations such as multiplying out brackets, are done explicitly, step by step.

Definition 5 (***PI-proofs***; System $\mathbb{P}_c(\mathbb{G})$, [8], [9])**.** *The system $\mathbb{P}_c(\mathbb{G})$ proves equations of the form $F = G$ over the ring $\mathbb{G}$, where F, G are algebraic circuits over $\mathbb{G}$. The inference rules of $\mathbb{P}_c$ are (with F, G, H ranging over all algebraic circuits, and where an equation below a line can be inferred from the one above the line):*

$$\text{R1}\quad \frac{F=G}{G=F} \qquad \text{R2}\quad \frac{F=G \qquad G=H}{F=H}$$

$$\text{R3}\quad \frac{F_1=G_1 \qquad F_2=G_2}{F_1+F_2=G_1+G_2} \qquad \text{R4}\quad \frac{F_1=G_1 \qquad F_2=G_2}{F_1\cdot F_2=G_1\cdot G_2}.$$

The axioms are equations of the following form, with F, G, H formulas:

A1	$F = F$
A2	$F + G = G + F$
A3	$F + (G + H) = (F + G) + H$
A4	$F \cdot G = G \cdot F$
A5	$F \cdot (G \cdot H) = (F \cdot G) \cdot H$
A6	$F \cdot (G + H) = F \cdot G + F \cdot H$
A7	$F + 0 = F$
A8	$F \cdot 0 = 0$
A9	$F \cdot 1 = F$
A10	$a = b + c\,,\ a' = b' \cdot c'$ *(if $a, b, c, a', b', c' \in \mathbb{G}$, are such that the equations hold in $\mathbb{G}$);*
A11	$F = F'$ *(when F, F' are* similar *circuits).*

A $\mathbb{P}_c$ proof *is a sequence of equations, called* ***proof-lines****, $F_1 = G_1$, $F_2 = G_2, \ldots, F_k = G_k$, with F_i, G_i circuits, such that every equation is either an axiom or was obtained from previous equations by one of the inference rules. The* ***size*** *of a proof is the total size of all circuits appearing in the proof. The* number of steps *in a proof is the number of proof-lines in it.*

A PI-proof can be easily verified for correctness in deterministic polynomial-time (assuming the field (or ring) has efficient representation; e.g., the field of rational numbers or the the ring $\mathbb{Z}$), simply by syntactically checking that each proof line is derived from previous lines by one of the inference rules.

F. Circuits and Proofs with Division

We denote by $\mathbb{G}(X)$ the field of formal rational functions in the variables X, where a formal rational fraction is a fraction of two formal polynomials with coefficients from $\mathbb{G}$. In this work we will consider $\mathbb{G}$ to be the ring of integers $\mathbb{Z}$. We will not be interested in 'inverse elements' in $\mathbb{Z}$ (excluding the element 1), nor in the completeness or soundness of proof systems for rational functions (like $\mathbb{P}_c^{-1}(\mathbb{Z})$ described below), because the theory will only prove *syntactical* properties of these proof systems (hence, no actual 'division' is performed over the integers.

It is possible to extend the notion of a circuit so that it computes rational functions in $\mathbb{G}(X)$ ([9]). This is done in the following way: a ***circuit with division*** F is an algebraic circuit which may contain an additional type of gate with fan-in 1, called an *inverse* or a *division* gate, denoted $(\cdot)^{-1}$. A division gate v^{-1} (i.e., a division gate whose incoming circuit is v) computes the rational function $1/\widehat{v} \in \mathbb{G}(X)$, assuming v does not compute the zero polynomial. If the circuit with division F contains some division gate v^{-1} such that v computes the zero polynomial, then we say that the circuit F is ***not well-defined***, and otherwise is ***well-defined***. Note, for instance, that the circuit $(x^2+x)^{-1}$ over $GF(2)$ is well-defined, since x^2+x is not the zero rational function (although it vanishes as a function over $GF(2)$).

We define the system $\mathbb{P}_c^{-1}(\mathbb{G})$, operating with equations $F = G$ where F and G are circuits with division [9], as follows: first, we extend the axioms of $\mathbb{P}_c(\mathbb{G})$ to apply to well-defined circuits with division. Second, we add the following new axiom:

$$\text{D} \qquad F \cdot F^{-1} = 1\,, \quad \text{provided that } F^{-1} \text{ is well-defined.}$$

IV. Carrying the Proof in the Theory

Here we describe in details how to prove the determinant identities in the theory, as highlighted before in Section II. We also explain where our construction in the theory differs from [9].

We assume all polynomials are over the ring of integers $\mathbb{Z}$. We reason inside $\mathbf{VNC}^2$ about $\mathbb{P}_c^{-1}(\mathbb{Z})$- and $\mathbb{P}_c(\mathbb{Z})$-proofs (Definition 5 and Sec. III-F). We use the following *reflection principle*, stating that if an equation has a proof then the equation is true:

Theorem 3 ($\mathbb{P}_c(\mathbb{Z})$-reflection principle; In $\mathbf{VNC}^2$)**.** *Let π be an $O(\log^2 n)$-depth $\mathbb{P}_c(\mathbb{Z})$-proof of the equation $F = G$. Then $F = G$* is true *in $\mathbb{Z}$ (that is, the $O(\log^2 n)$-depth algebraic circuits F and G compute the same function over the integers).*

Theorem 3 is proved as follows. We define the *evaluation function* for $O(\log^2 n)$-depth algebraic circuits over $\mathbb{Z}$ as the function that receives an integer assignment A and an $O(\log^2 n)$-depth algebraic circuit C. The algorithm then converts C into a *Boolean* $\mathbf{NC^2}$ circuit, where the inputs are the bit-strings corresponding to A. And then evaluates the Boolean circuit using evaluation of $\mathbf{NC^2}$ circuits ($\mathbf{\Sigma}_1^B$-definable in $\mathbf{VNC}^2$), and finally outputs the result.

We also need to show in $\mathbf{VNC}^2$ that the rules and axioms of $O(\log^2 n)$-depth $\mathbb{P}_c(\mathbb{Z})$ are sound with respect to the above evaluation function. This is proved by inspection of each of the axioms and rules.

f) The determinant function DET *(in the theory):* The (uniform-$\mathbf{NC^2}$) determinant function DET is defined in the theory via the algorithm below. Each step in the algorithm corresponds to a (more involved) step in the algorithm that constructs the final PI-proof of the determinant identities in the theory. We will defer the more detailed explanation of each step in the algorithm to the sequel, in which we explain the corresponding steps of the PI-proof construction.

Algorithm DET (in $\mathbf{VNC}^2$)
Input: an $n \times n$ integer matrix A.
Output: $z \in \mathbb{Z}$, where z is the determinant of A.

1) Write down an unbalanced algebraic circuit $\text{Det}_{circ^{-1}}(X)$ with division gates that computes the symbolic $n \times n$ determinant polynomial, over the variables $X = \{x_{ij}\}_{i,j\in[n]}$. This circuit captures the standard recursive block-wise formula for computing the determinant of matrices, using "Schur complement" (intuitively, it captures the Gaussian elimination procedure). For details see Sec. VI-A.
2) Consider the circuit $\text{Det}_{circ^{-1}}(I_n + zX)$ as computing a univariate polynomial in the new variable z. Using this circuit, construct a new circuit $\text{Det}_{Taylor}(X)$ computing the nth term of the Taylor expansion of $\text{Det}_{circ^{-1}}(I_n + zX)$ around $z = 0$.
3) Convert the circuit $\text{Det}_{Taylor}(X)$ into a circuit that has a syntactic-degree n, denoted $\text{Det}'_{Taylor}(X)$.
4) Convert Det'_{Taylor} into a circuit with a single division gate at the top, denoted Det''_{Taylor}.
5) Eliminate the division gate u^{-1} from Det''_{Taylor} by substituting u with a truncated power series of u^{-1} around a point defined by the identity matrix. Denote the new circuit by Det_{circ}.
6) Balance Det_{circ} via the (uniform) balancing algorithm. Denote by $\text{Det}_{balanced}$ the resulting $O(\log^2 n)$-depth and $\text{poly}(n)$-size circuit.
7) Evaluate the circuit $\text{Det}_{balanced}$ with the input assignment A, using the evaluation function as defined above.

Since we show that all the constructions above are Σ_1^B-definable functions in $\mathbf{VNC}^2$, the determinant function as defined above is Σ_1^B-definable in the theory (namely, totally recursive).

Given the function DET we now sketch the proof in $\mathbf{VNC}^2$ of the two equations (1), (2) above.

Existence of proofs with division gates $\mathbb{P}_c^{-1}(\mathbb{Z})$. We show in $\mathbf{VNC}^2$ the existence of a function (i.e., a $\mathbf{\Sigma}_1^B$-definable function) that given a number n in unary outputs a $\mathbb{P}_c^{-1}(\mathbb{Z})$-proof π_0 of equations (1) and (2) (these are equations between algebraic circuits over $\mathbb{Z}$). This is a proof in which circuits have exponential syntactic-degrees (though the theory cannot express this fact). The circuits in the proof are not necessarily homogeneous, and have division gates. Note that $\text{Det}_{circ^{-1}}(X)$ computes the determinant as a rational function (and not as a polynomial).

The determinant as a polynomial. Let $\text{Det}_{Taylor}(X)$ be the circuit computing the nth term of the Taylor expansion of $\text{Det}_{circ^{-1}}(I_n + zX)$ around $z = 0$. We argue that the ("inverse") ring element needed to be used for this Taylor expansion is the element 1 (and thus it has an inverse in $\mathbb{Z}$).

The circuit $\text{Det}_{Taylor}(X)$ thus computes the determinant function (intuitively, since z multiplies every variable x_{ij}), and by construction it will have no division gates. Hence, it computes the determinant *as a polynomial*. We show that $\mathbf{VNC}^2$ proves the existence of a function that given a number n in unary outputs a $\mathbb{P}_c^{-1}(\mathbb{Z})$-proof of $\text{Det}_{Taylor}(X) = \text{Det}_{circ^{-1}}(X)$. Thus, combined with the previous part, $\mathbf{VNC}^2$ proves the existence of a $\mathbb{P}_c^{-1}(\mathbb{Z})$-proof, denoted π_1, of the determinant identities (1), (2), where the determinant DET is now replaced by Det_{Taylor} in the identities.

Reducing the syntactic-degree of the determinant polynomial. The circuit $\text{Det}_{Taylor}(X)$ has exponential syntactic-degree[3]. However, for the next step, we need $\text{Det}_{Taylor}(X)$ to have a polynomial syntactic-degree. We show in $\mathbf{VNC}^2$, that there exists a $\mathbb{P}_c^{-1}(\mathbb{Z})$-proof of $\text{Det}_{Taylor}(X) = \text{Det}'_{Taylor}(X)$, where $\text{Det}'_{Taylor}(X)$ has syntactic-degree n.

Thus, by previous parts, $\mathbf{VNC^2}$ proves the existence of a $\mathbb{P}_c^{-1}(\mathbb{Z})$-proof of the determinant identities (1), (2), where the determinant DET is now replaced by Det'_{Taylor} which is an algebraic circuit with no division gates and of syntactic-degree n. Denote this proof by π_2.

Bringing division gates to the top (Shown in details below; Sec. V-A). We say that a circuit C *has a division at the top* whenever C is of the form $F \cdot (G)^{-1}$ or $(G)^{-1} \cdot F$, for two circuits F, G. If F, G do not have division gates we say that C *has a single division gate at the top*. We need our circuits to have such a structure, because if we have circuits with nested divisions we cannot replace division gates by an "approximating" power series in the next step.

We devise an $\mathbf{NC^2}$ algorithm that takes an algebraic circuit with division, *of any depth*, and outputs an algebraic circuit

[3]Here, we shall differ from [9], since we do not know how to formulate an $\mathbf{NC^2}$-algorithm for eliminating 0 nodes in general algebraic circuits.

computing the same rational function that has a *single* division gate at the top of the circuit, i.e., the root (this is a slight abuse of notation; see Sec. V-A).

This algorithm is not entirely trivial due to the need to work in $\mathbf{NC^2}$. We moreover show that this algorithm is $\mathbf{\Sigma}_1^B$-definable in $\mathbf{VNC}^2$.

Then, using this algorithm, we show in $\mathbf{VNC}^2$ how to convert the $\mathbb{P}_c^{-1}(\mathbb{Z})$*-proof* π_2 into a proof in which every circuit has a single division gate at the top. Denote the resulted proof by π_3

Eliminating division gates. We now wish to eliminate the division gates from the $\mathbb{P}_c^{-1}(\mathbb{Z})$-proofs, to obtain $\mathbb{P}_c(\mathbb{Z})$-proofs without divisions. Standard division elimination by Strassen [16] requires finding a (total) assignment to the variables, such that no division gate in the circuit equals zero under this assignment. However, we do not know how to uniformly find such assignments, and so we do not know how to uniformly eliminate division gates from general algebraic circuits in $\mathbf{VNC}^2$. Our division elimination will work only for those circuits **in** π_3.

First, we show that the assignment of identity matrices to the (matrix) variables $A = \{a_{ij}\}, B = \{b_{ij}\}, C = \{c_{ij}\}$ $(i, j \in [n])$, in the proof π_3 of equations (1), (2) does not nullify any division gate in π_3 (though this statement is not expressed in the theory).

Assuming for simplicity that w_i (for $i \in J$) are all the variables in π_3 and let b be the assignment of identity matrices to the variables in π_3. Then, substitute in π_3 the term $(b_i - y_i)$ for each w_i (for all $i \in J$) denoting the obtained proof by π_3'. Then the all zero assignment $\overline{0}$ to the y_i variables in π_3' does not nullify any division gate in π_3'. Furthermore, we show that under this assignment every division gate computes the polynomial 1 (and thus has an inverse in $\mathbb{Z}$). Therefore, in the theory, we simply construct this (substitution instance) $\mathbb{P}_c^{-1}(\mathbb{Z})$-proof π_3' (though, again, we do not express in the theory the argument just discussed).

Let $\mathrm{Inv}_k(H)$ be the truncated power series of H^{-1} over the point determined by the identity matrix (loosely speaking, serving as the inverse polynomial of H "up to the kth power"). Specifically, $\widehat{H} \cdot \widehat{\mathrm{Inv}_k(H)} = 1 + [\text{terms of degree} > k]$.[4]

For every circuit C with a top division gate H^{-1}, $\mathbf{VNC}^2$ proves there exists a corresponding division-free circuit C', obtained by replacing the division gate H^{-1} in C by $\mathrm{Inv}_k(H)$.

Let π_4 be the corresponding division-free proof-sequence obtained from π_3' by replacing every circuit with the corresponding division-free circuit as above. By itself π_4 is not a legal $\mathbb{P}_c(\mathbb{Z})$-proof, since the axiom of division in $\mathbb{P}_c^{-1}(\mathbb{Z})$ does not translate into an axiom in $\mathbb{P}_c(\mathbb{Z})$. In other words, the axiom of division D, $F \cdot F^{-1} = 1$ (provided that F^{-1} is defined[5]; see Definition 5), translates into $F \cdot \mathrm{Inv}_n(F) = 1 + [\text{terms of degree} > n]$. We fix this problem in the next step.

[4]Though, again, $\mathbf{VNC}^2$ cannot prove this equality, since general evaluation of (unrestricted depth) algebraic circuits is not known to be defined in the theory.

[5]F^{-1} is *defined* whenever the polynomial computed by any division gate is nonzero.

Eliminating high degrees. Here we eliminate the high syntactic-degrees ($> n$) parts in the circuits appearing in π_4. This is done by homogenizing the proof π_4. Specifically, we show in $\mathbf{VNC}^2$ the existence of a function that receives an algebraic circuit G of syntactic-degree k and converts it into a sum of $k+1$ syntactic-homogeneous circuits $\sum_{i=0}^{k} G^{(i)}$ (computing the same polynomial), where $G^{(i)}$ denotes a syntactic-homogeneous circuit of syntactic-degree i computing the sum of all degree i monomials in G.

Moreover, we show that $\mathbf{VNC}^2$ can prove the existence of a function that given a $\mathbb{P}_c(\mathbb{Z})$-proof of a syntactic-degree n equation $F = G$, decomposes the proof into $n+1$ $\mathbb{P}_c(\mathbb{Z})$-proofs of $F^{(i)} = G^{(i)}$, for $i = 0, \dots, n$, each proof having syntactic-degree at most i. Combining these proofs gives a low syntactic-degree version of π_4.

This also fixes the problem caused by division elimination described at end of the previous step. We thus obtain a $\mathbb{P}_c(\mathbb{Z})$-proof, denoted π_5, of (the degree n syntactic-homogeneous parts of) equations (1) and (2).

Balancing algebraic circuits is definable in the theory. This follows the algorithm discussed in Sec. V-B. Moreover we show in $\mathbf{VNC}^2$ the existence of a function that receives a $\mathbb{P}_c(\mathbb{Z})$-proof of $F = G$ with syntactic-degree d, and outputs a $\mathbb{P}_c(\mathbb{Z})$-proof of $[F] = [G]$ in which every circuit is of depth-$O(\log s \cdot \log d + \log^2 d)$ and the size of the proof is $\mathrm{poly}(s, d)$.

Applying this function to π_5, we obtain a $\mathbf{\Sigma}_1^B$-definable function in $\mathbf{VNC}^2$, that given n in unary outputs a depth-$O(\log^2 n)$ $\mathbb{P}_c(\mathbb{F})$-proof π_6 of the determinant identities (1), (2) (where DET is replaced by the appropriate balanced circuit computing the determinant, denoted $\mathsf{Det}_{balanced}$) .

Applying the reflection principle. We now reason in $\mathbf{VNC}^2$ as follows: for every n and every pair of matrices A, B over $\mathbb{Z}$ of dimension $n \times n$, by the definition of DET, DET(AB), DET(A) and DET(B) equals the value of applying the evaluation function to the circuit $\mathsf{Det}_{balanced}$ with the input assignment AB, A, B, resp. (where the matrix product AB is definable in $\mathbf{VNC}^2$; cf. [6]).

By the arguments above, there exists a depth-$O(\log^2 n)$ $\mathbb{P}_c(\mathbb{Z})$-proof of $\mathsf{Det}_{balanced}(XY) = \mathsf{Det}_{balanced}(X) \cdot \mathsf{Det}_{balanced}(Y)$ for the two symbolic matrices X, Y of dimension $n \times n$. But by Theorem 3 this means that for every input matrices over $\mathbb{Z}$, $\mathsf{Det}_{balanced}(AB) = \mathsf{Det}_{balanced}(A) \cdot \mathsf{Det}_{balanced}(B)$. Therefore, by the above, DET(AB) = DET$(A) \cdot$ DET(B). Similar reasoning applies to the proof of determinant identity (2).

V. The Uniform $\mathbf{NC^2}$ Algorithms

Here we describe some of the uniform $\mathbf{NC^2}$-algorithms we develop for the construction of the PI-proofs in the theory and for proving the soundness of PI-proofs in the theory. In particular, we focus on *division normalization* of both circuits and proofs—namely, converting an algebraic circuit (PI-proof, resp.) with division gates into a circuit with only one division gate at *the top*, i.e., at the output gate (PI-proof in which every circuit has division only at the top, resp.). One reason we focus on this construction here, is that both homogenization of proofs (and circuits) and balancing of proofs (and circuits) in the theory follows to a certain extent the division normalization scheme we describe here. We then describe in general terms the $\mathbf{NC^2}$-algorithms for balancing circuits.

Further $\mathbf{NC^2}$-constructions that we skip due to lack of space are breaking circuits (and proofs) into their homogeneous components (the standard Strassen's [16] algorithm lends itself quite immediately to a parallel execution, but constructing the homogenized *proofs* needs some care), and the $\mathbf{NC^2}$-algorithm for the *algebraic-*$\mathbf{NC^2}$ *circuit evaluation problem* (over $\mathbb{Z}$).

A. Parallel Division Normalization of Circuits and Proofs

Here we show the parallel algorithm that receives an algebraic circuit with division gates and normalizes it, that is, converts it into a circuit with a single division gate at the top (i.e., output gate), and similarly for $\mathbb{P}_c^{-1}$-proofs. For simplicity, we shall sometimes abuse notation and assume in this section that the division gates has fan-in *two*, so that a circuit with a division gate at the top can be written as $F \div G$, where $\div$ is a division gate.

For every node v in a circuit F with division introduce two nodes $\mathrm{Den}(v)$ and $\mathrm{Num}(v)$ that will compute the numerator and denominator of the rational function computed by v, respectively, as follows:

1) If v is an input node of F, let $\mathrm{Num}(v) := v$ and $\mathrm{Den}(v) := 1$.
2) If $v = u^{-1}$, let $\mathrm{Num}(v) := \mathrm{Den}(u)$ and $\mathrm{Den}(v) := \mathrm{Num}(u)$.
3) If $v = u_1 \cdot u_2$, let $\mathrm{Num}(v) := \mathrm{Num}(v_1) \cdot \mathrm{Num}(v_2)$ and $\mathrm{Den}(v) := \mathrm{Den}(v_1) \cdot \mathrm{Den}(v_2)$.
4) If $v = u_1 + u_2$, let $\mathrm{Num}(v) := \mathrm{Num}(u_1) \cdot \mathrm{Den}(u_2) + \mathrm{Num}(u_2) \cdot \mathrm{Den}(u_1)$ and $\mathrm{Den}(v) := \mathrm{Den}(u_1) \cdot \mathrm{Den}(u_2)$.

Let $\mathrm{Num}(F)$ and $\mathrm{Den}(F)$ be the circuits with the output node $\mathrm{Num}(w)$ and $\mathrm{Den}(w)$, respectively, where w is the output node of F. We want to show the following:

Theorem 4 (in $\mathbf{VNC}^2$)**.** *(i) If F is a circuit with division, then $F = \mathrm{Num}(F) \cdot \mathrm{Den}(F)^{-1}$ has a $\mathbb{P}_c^{-1}(\mathbb{F})$ proof. (ii) Let F, G be circuits with division. Assume that $F = G$ has a $\mathbb{P}_c^{-1}(\mathbb{F})$ proof. Then $\mathrm{Num}(F) \cdot \mathrm{Den}(F)^{-1} = \mathrm{Num}(G) \cdot \mathrm{Den}(G)^{-1}$ has a $\mathbb{P}_c^{-1}(\mathbb{F})$ such that every division gate in every circuit in the proof occurs only at the top.*

We prove only part (i), that exemplifies the main idea. To prove this we first describe the $\mathbf{NC^2}$-algorithm that normalizes circuits with divisions, as follows (we ignore encoding issues):

$\mathbf{NC^2}$*-Algorithm for Normalizing Circuits with Divisions*

<u>*Input*</u>: C an algebraic circuit with division gates.
<u>*Output*</u>: An algebraic circuit computing $\widehat{C}$ with a single division gate at the top.

1) Convert C into a *layered* algebraic circuit C'. This can be done in $\mathbf{NC^1}$ (we skip this procedure due to lack of space).
2) (*Sequentially*) **For** every $i = \lceil\log(d)\rceil, \dots, 2, 1$, where d is the depth of C (starting with $i = \lceil\log(d)\rceil$), **do**:
   a) Consider the (layered) circuit as divided into 2^i blocks. (A block thus contains all the subcircuits whose roots are at the top of the block and leaves are at the bottom of the block.)
      In parallel, for each *pair* of consecutive blocks, **do**:
      ★ (At this stage, each block possibly contains division gates only at its top.) Move all division gates in the top of the lower block to the top of the upper block.

Step (a) in the algorithm above ends with all division gates occurring at the top of the upper block of each of the pairs considered.

Since the above algorithm has $O(\log d)$ steps, to conclude that the above algorithm is in $\mathbf{NC^2}$, it suffices to show that step ★ can be implemented in $\mathbf{NC^1}$:

$\mathbf{NC^1}$-algorithm for moving all division gates in the top of a lower block to the top of an upper block

<u>*Input*</u>: C a layered algebraic circuit with division gates, partitioned into two halves: an *upper block* consisting of the layers in the upper half and a *lower block* consisting of the layers in the lower half, where division gates may occur only in the top layer of each block.
<u>*Output*</u>: An algebraic circuit with division computing $\widehat{C}$ with all division gates at the top of the upper block.

1) Syntactically multiply <u>*all*</u> nodes in C (in both blocks) by the *product* α of *all* denominators α_j occurring in the top level of the lower block (as $\beta_j \div \alpha_j$, for some β_j).
2) Cancel accordingly the denominators of all top-layer nodes in the lower block, so that now all gates in lower and upper blocks have *no denominators*, except for the top layer nodes in the upper block.
3) Add a denominator α (i.e., syntactically divide by the sub-circuit α) to all the gates in the top layer of the upper block. It is easy to check that the new circuit we get computes $\widehat{C}$.

Notes on the above algorithm: when adding products like α we just add edges to a single sub-circuit computing α. When we add edges in the above algorithm we always preserve the circuit *being layered* (so we may need to add sufficiently many dummy edges to preserve the "layerness" of the circuit).

We now turn to the proof of Theorem 4 (i).

Proof sketch of Theorem 4 (i). In the "$\mathbf{NC^2}$*-Algorithm for Normalizing Circuits with Divisions*" we had $\lceil\log d\rceil$ steps, for d the depth of the input circuit. Similarly, we describe an $\mathbf{NC^2}$-algorithm for constructing the $\mathbb{P}_c^{-1}(\mathbb{Z})$-proof of $F = \mathrm{Num}(F) \div \mathrm{Den}(F)$.

In each step $i = \lceil\log d\rceil, \dots, 2, 1$, where d is the depth of F, we construct (in parallel) a $\mathbb{P}_c^{-1}(\mathbb{Z})$-proof for the correctness of step ★, for every pair of consecutive blocks in F, using:

Claim 5 (in $\mathbf{VNC}^2$; in fact in $\mathbf{VNC}^1$)**.** *Let C, C' be two layered circuits with division gates, of depth k each. Assume* *that C' is the result of applying step* ★ *in the division normalization above on circuit C. Then there is a $\mathbb{P}_c^{-1}(\mathbb{Z})$-proof of $C = C'$.*

We omit the proof of this claim. □

B. Balancing Algebraic Circuits (and Proofs) in Uniform-$\mathbf{NC^2}$ and in $\mathbf{VNC}^2$

Here we provide some overview of the $\mathbf{NC^2}$-algorithm for constructing the balanced circuit, given as an input an upper bound (in unary) on the syntactic-degree of the input circuit. Due to lack of space we focus only on some of the differences between our algorithm and the standard Valiant *et al.* [17] algorithm.

$\mathbf{NC^2}$-algorithm for balancing circuits (overview)

Input: C, d where C is a syntactic-homogeneous circuit and d is the syntactic-degree of C in unary (we can assume that C is a syntactic-homogeneous circuit because we can transform in parallel a circuit into a syntactic-homogeneous circuit; we can also assume that we get d as an input because of this homogenization algorithm).

Output: A balanced circuit $[C]$ computing the polynomial $\widehat{C}$. That is, if C has size s, then the depth of $[C]$ is $O(\log s \log d + \log^2 d)$ and the size of $[C]$ is $\text{poly}(s, d)$.

Algorithm: The algorithm proceeds via the general scheme of [17] in $\lceil \log d \rceil$ stages, combining it with a case of the Miller *et al.* [12] algorithm (the Miller *et al.* algorithm is an $\mathbf{NC^2}$-algorithm for *evaluating* an algebraic circuit under an assignment).

We list some of the specific features of our $\mathbf{NC^2}$ balancing algorithm:

1) The notion of degree in the original construction [17] is replaced in our algorithm with that of a *syntactic-degree*. Because neither syntactic-degrees (nor degrees) can be computed (apparently) in $\mathbf{NC^2}$, we need to supply it as an input to the algorithm; the syntactic-degree d is used in the algorithm, because we balance the circuit in $\lceil \log d \rceil$ stages.

2) Another difference between our construction and [17] is implied by the need to evaluate *unbalanced* constant algebraic circuits (i.e., circuits with no variables) in the original construction. Specifically, in the base case of our construction, when we are given a circuit computing a linear form we need to compute the coefficients of the linear form. However, we cannot directly compute these coefficients since the (variable-free) circuit computing the linear form may be a circuit beyond $\mathbf{NC^2}$ (e.g., a circuit of linear depth) and so it may not be possible to directly evaluate it within $\mathbf{NC^2}$.

One way of solving this problem is to replace each field element $c \in \mathbb{Z}$ that occurs in the circuit with a new variable x_c, and by that making sure that all sub-circuits computing linear forms will contain only variables; and hence, there will remain *no* unbalanced variable-free sub-circuits in the circuit (when we balance all syntactic-degree 1 sub-circuits in the circuit). For this replacement to be useful we need to make sure that the syntactic-degree of the circuits obtained after the replacement of ring elements by new variables is still polynomial in n.

When we compute circuits of syntactic-degree 1 in the base case of the construction, there will be no scalars in the circuits (since we replaced scalars by variables in advance), namely, circuits of syntactic-degree 1 will contain only plus gates. We thus need to evaluate arithmetic circuits with only plus gates (note that the circuits are not necessarily balanced). As mentioned above, to do this in $\mathbf{NC^2}$ we follow a similar approach to that in [12].

VI. Encoding Circuits and Proofs: The Determinant Circuit in the Theory

Here we give some details on how to encode and construct the required circuits (and proofs) in $\mathbf{VNC}^2$. We focus on the already non-trivial construction of the determinant circuit with division $\text{Det}_{circ^{-1}}$ in the theory; encoding and constructing proofs in the theory follow similar lines.

A. Circuit with Division for the Determinant

First we need to define the determinant *circuit with division* denoted $\text{Det}_{circ^{-1}}$. Similar to [9], this is done using block-wise inversion: by considering the symbolic matrix $X = \{x_{ij}\}_{i,j\in[n]}$, consisting of n^2 distinct variables, defining the matrix inverse X^{-1} of X and then, by partitioning X into blocks, we formulate a recursive definition of the determinant, using matrix inverse. This definition can be viewed as a formulation of Gaussian elimination.

Specifically, we define an $n \times n$ matrix X^{-1} whose entries are circuits with divisions, computing the inverse of X, as follows:

1) If $n = 1$, let $X^{-1} := (x_{11}^{-1})$.
2) If $n > 1$, write X as follows:
$$X = \begin{pmatrix} X_1 & v_1^t \\ v_2 & x_{nn} \end{pmatrix}, \tag{6}$$
where $X_1 = \{x_{ij}\}_{i,j\in[n-1]}$, $v_1 = (x_{1n}, \ldots, x_{(n-1)n})$ and $v_2 = (x_{n1}, \ldots, x_{n(n-1)})$. Assuming we have constructed X_1^{-1}, let
$$\delta(X) := x_{nn} - v_2 X_1^{-1} v_1^t\,. \tag{7}$$
$\delta(X)$ computes a single non-zero rational function and so $\delta(X)^{-1}$ is well-defined. Finally, let
$$X^{-1} := \begin{pmatrix} X_1^{-1}\left(I_{n-1} + \delta(X)^{-1} v_1^t v_2 X_1^{-1}\right) & -\delta(X)^{-1} X_1^{-1} v_1^t \\ -\delta(X)^{-1} v_2 X_1^{-1} & \delta(X)^{-1} \end{pmatrix} \tag{8}$$

The circuit $\text{Det}_{circ^{-1}}(X)$ is defined as follows (using "Schur complement"):

1) If $n = 1$, let $\text{Det}_{circ^{-1}}(X) := x_{11}$.
2) If $n > 1$, partition X as in (6) and let $\delta(X)$ be as in (7). Let $\text{Det}_{circ^{-1}}(X) :=$
$$\text{Det}_{circ^{-1}}(X_1) \cdot \delta(X) = \text{Det}_{circ^{-1}}(X_1) \cdot (x_{nn} - v_2 X_1^{-1} v_1^t)\,.$$

The definition in (8) should be understood as a circuit with n^2 outputs which takes $X_1^{-1}, v_1, v_2, x_{nn}$ *as inputs* and moreover, such that the inputs from X_1^{-1} *occur exactly once*. Altogether, we obtain a polynomial-size circuit for X^{-1} and the determinant function of X. The circuits obtained are unbalanced, have division gates and are of exponential syntactic-degree (see Definition 4). The fact that $\mathsf{Det}_{circ^{-1}}(X)$ indeed computes the determinant (as a rational function) stems, e.g., from the fact (shown in this work, or in [9]) that $\mathbb{P}_c^{-1}(\mathbb{Z})$ can prove the two identities that characterize the determinant. That X^{-1} computes the matrix inverse is also proved in the theory.

B. Constructing the Circuit $\mathsf{Det}_{circ^{-1}}$ *in* $\mathbf{V}^0$

Here we show a $\mathbf{\Sigma}_0^B$-definable string function in $\mathbf{VNC}^2$ (in fact in $\mathbf{V}^0$), denoted $\mathsf{write}_{X^{-1}}$, that outputs the multi-output circuit X^{-1} ((8) above) given as input a unary integer n (among other parameters).

Unlike (8), the function $\mathsf{write}_{X^{-1}}$ is *not recursive*, as the circuit is of depth $\Omega(n)$ and we do not have in $\mathbf{V}^0$, nor in $\mathbf{VNC}^2$, the induction axiom for Σ_1^B-formulas. Fortunately, we need the theory only to construct the circuit syntactically.

The circuit for X^{-1} is encoded as follows. It is a multi-output circuit. The string V encodes the nodes in the circuit. For every layer $d = 1, \dots, n$ in the inductive definition of X^{-1}, we have a set of nodes $(d,(i,j),\ell) \in V$, where (i,j), for $i,j \in [d]$, is an entry in a $d \times d$ matrix, meaning that the node $(d,(i,j),\ell)$ is part of a sub-circuit of X^{-1} that computes the (i,j)th entry in the dth inductive-step; ℓ is the running index of the nodes in that part, where $\ell = 0$ iff the node is what we consider an *output node of the given* d *and the given entry* (i,j). Nodes of the form $(0,(i,j),0)$ stand for the *input variable* x_{ij} of the matrix X; therefore, these are the input variables of the circuit X^{-1}.

For example, $(1,(1,1),0)$ is the node computing x_{11}^{-1}, because the first coordinate $d = 1$ refers to the "recursive" level 1 in (8) above, the second is $(1,1)$, meaning the $(1,1)$-entry from the circuit computing the inverse of x_{11}, and the last coordinate is 0, meaning this is the *output* node of the inverse of x_{11}.

Additionally, we have a string G encoding the gate-type of each node in V, excluding the input nodes $(0,(i,j),0)$. That is, $(d,(i,j),\ell,g) \in G$ means that node $(d,(i,j),\ell) \in V$ is of type $+$ if $g = 0$, $\times$ if $g = 1$ and division $\div$ if $g = 2$, and an input variable x_{ij} if $g = \langle i,j \rangle$, where $\langle \cdot,\cdot \rangle$ is the pairing function (note that the pairing function is monotone increasing and that $\langle 1,1 \rangle > 2$, so we can distinguish between the case of an arithmetic gate and an input gate). Finally, the string E encodes the edges between nodes in the circuit. That is, $(d,(i,j),\ell,d',(i',j'),\ell')$ means that there is a directed edge from node $(d,(i,j),\ell)$ to node $(d',(i',j'),\ell')$.

Using the above encoding scheme it is possible now to bit-define the string function $\mathsf{write}_{X^{-1}}$ as a $\mathbf{\Sigma}_0^B$-definable function in $\mathbf{VNC}^2$. We only need to construct, given some fixed $d,(i,j)$, the sub-circuits whose nodes will be $(d,(i,j),\ell)$, for some ℓ, according to the definition in (8). We will use the following notation and functions in the theory.

Notations and basic functions for constructing sub-circuits Let F be some "simple" arithmetic function, such as inner product of two n-element vectors over the integers, or one of the functions in (8) used to define a minor or the matrix inverse X^{-1}, such as $\delta(X)^{-1}$. We will denote by $\mathsf{write}_F(n,d,\ell,\overline{I},\overline{O})$ the following string function: the inputs are $\overline{I}$, serving as the input nodes to the circuit and $\overline{O}$ the output nodes of the circuit for F, d is the index "level" (used to record the recursive level of recursive circuit constructions as in (8)) and ℓ is the "running" index of a node in a given level d, and n stands for the "dimension" of the operation defined by F (e.g., inner product of vectors of size n, or matrix product of two $n \times n$ matrices has dimension n). The output is a string, but we abuse notation and assume it is *three* separate strings encoding the (output) circuit, for simplicity, as follows: E, V, G as described above.

More formally, we define $\mathsf{write}_F(n,d,\ell,\overline{I},\overline{O}) = (E,V,G)$ as follows (similar to the above notation). V is a string describing the vertices in an algebraic circuit. E is a string describing the edges between vertices in V. G is a string describing the gate-types of vertices in V. Every vertex is of the form $(d,(i,j),\ell)$ with d the recursive level in the definition of X^{-1} in (8), (i,j) means that the node is in the (i,j)'s part of the definition of X^{-1}, and ℓ is the running index of nodes in the same level d and same part (i,j), where $\ell = 0$ iff the node is an output node of *that level* d (it is not necessarily the output node of the whole circuit). Assume that $F(\overline{I})$ is some algebraic function with m_0 integer inputs $\overline{I}$ and m_1 integer outputs $\overline{O}$. Then, we supply $\mathsf{write}_F(n,d,\ell,\overline{I},\overline{O})$ with the nodes indices (as encoded in V) to be used as input nodes and output nodes for the (sub-)circuit computing F.

Example: Consider $F_1 := X_1^{-1}(I_{n-1} + \delta(X)^{-1} v_1^t v_2 X_1^{-1})$ from (8). This is a recursive function in the sense that it uses the output X_1^{-1} which is computed in the previous recursive level $d-1$ as input, together with the "new" nodes in row d and column d in X. Therefore, the inputs of F_1 are the following nodes: $(d-1)^2$ input nodes for X_1^{-1}, $2(d-1)$ input nodes for v_1^t and v_2, and finally one input node x_{dd} (needed for computing $\delta(X)^{-1}$), which sums up to d^2 input nodes. The number of output nodes for F_1 is $(d-1)^2$, as it defines a $(d-1) \times (d-1)$ minor of X^{-1}. Thus, in our encoding scheme, the input nodes (viewed as a $d \times d$ matrix) are:

$$\begin{pmatrix} (d-1,(1,1),0) & \dots & (d-1,(1,d-1),0) & (0,(1,d),0) \\ \vdots & \ddots & & \vdots \\ (d-1,(d-1,1),0) & \dots & (d-1,(d-1,d-1),0) & (0,(d-1,d),0) \\ (0,(d,1),0) & \dots & (0,(d,d-1),0) & (0,(d,d),0) \end{pmatrix}$$

and the output nodes (viewed as a $(d-1) \times (d-1)$ matrix) are:

$$\begin{pmatrix} (d,(1,1),0) & \dots & (d,(1,d-1),0) \\ \vdots & \ddots & \vdots \\ (d,(d-1,1),0) & \dots & (d,(d-1,d-1),0) \end{pmatrix}.$$

Let F_1, F_3, F_4 be the other three functions used in the definition of X^{-1} (8) (for the other three minors). We can define similarly write_{F_i} functions for these F_i's.

To actually show that $\mathsf{write}_{X^{-1}}$ is a $\mathbf{\Sigma}_0^B$-definable function in $\mathbf{V}^0$ we need to show, e.g., how to bit-define $\mathsf{write}_{v\cdot u}$ using a $\mathbf{\Sigma}_0^B$-formula, given two n-element vectors of integers v, u representing *nodes* in the circuit. This is quite easy to show: simply output a binary tree with the correct plus and products nodes, and plug the input nodes v, u to the leaves accordingly.

Similarly, we have $\mathbf{\Sigma}_0^B$-formulas for constructing other formulas like write_{vA} and write_{Av^t}, given the input nodes for an $n \times n$ matrix A, and the input nodes for an n-elements vector v. Similarly, given a node z it is trivial to output a circuit computing z^{-1} or $-z$, and given two matrices A, B (i.e., $2n^2$ nodes) it is trivial to $\mathbf{\Sigma}_0^B$-define write_{A+B} in $\mathbf{V}^0$.

Now that we set up the notation and the functions for constructing sub-circuits, we can $\mathbf{\Sigma}_0^B$-define $\mathsf{write}_{X^{-1}}$ in $\mathbf{V}^0$ as follows. First, for $i = 1, \ldots, 4$, define $\mathrm{Inp}_{F_i}(d)$ and $\mathrm{Out}_{F_i}(d)$ as the string functions that output the sequence of input- (output-, respectively) nodes of the dth recursive level of X^{-1} for each of the F_i's, as shown for F_1 in the example above. They are all definable string-functions in $\mathbf{VNC}^2$. We can now bit-define $\mathsf{write}_{X^{-1}}$ as follows:

$$\mathsf{write}_{X^{-1}}(n, \ell, \overline{I}, \overline{O})(i) \equiv$$
$$\exists 2 \le d \le n \left(\mathsf{write}_{\mathsf{level}(X^{-1})}\left(n, d, 1, \mathrm{Inp}_{F_i}(d), \mathrm{Out}_{F_i}(d)\right)(i)\right) \wedge$$
$$\mathsf{write}_{x_{11}^{-1}}\left(n, 1, 0, ((0, (1,1), 0)), ((1, (1,1), 0))\right)(i)\Big),$$

where $\mathsf{write}_{\mathsf{level}(X^{-1})}(n, d, \ell, \overline{I}, \overline{O})$ outputs (E, V, G) encoding a (sub-)circuit that is the dth inductive level of X^{-1}, and $\mathsf{write}_{x_{11}^{-1}}\left(n, 1, 0, ((0, (1,1), 0)), ((1, (1,1), 0))\right)$ is the string function that outputs the encoding of the circuit "x_{11}^{-1}".

VII. Conclusions

We establish a uniform proof, in what may be considered the weakest logical theory possible, of the basic determinant identities. This answers an open question of, e.g., [4]. We achieve this by formalizing in the theory $\mathbf{VNC}^2$ the construction of the propositional proofs from Hrubeš-Tzameret [9], and using a reflection principle for PI-proofs in the theory, devising along the way parallel ($\mathbf{NC^2}$) algorithms for basic algebraic-circuit constructions, provably total in $\mathbf{VNC}^2$. Due to the central role of linear algebra and the determinant function, these results are expected to be relevant to further basic work in bounded arithmetic.

As for the $\mathbf{VNC}^2$-provability of the *Cayley-Hamilton* theorem and the *co-factor expansion* of the determinant, we believe that these should follow relatively easy from our results.

Acknowledgements

We thank Pavel Hrubeš for useful discussions related to [9].

References

[1] Paul Beame and Toniann Pitassi. Propositional proof complexity: past, present, and future. *Bull. Eur. Assoc. Theor. Comput. Sci. EATCS*, (65):66–89, 1998.

[2] Maria Luisa Bonet, Samuel R. Buss, and Toniann Pitassi. Are there hard examples for Frege systems? In *Feasible mathematics, II (Ithaca, NY, 1992)*, volume 13 of *Progr. Comput. Sci. Appl. Logic*, pages 30–56. Birkhäuser Boston, Boston, MA, 1995.

[3] Samuel R. Buss. *Bounded Arithmetic*, volume 3 of *Studies in Proof Theory*. Bibliopolis, 1986.

[4] Stephen Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity*. ASL Perspectives in Logic. Cambridge University Press, 2010.

[5] Stephen A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64(1-3):2–21, 1985.

[6] Stephen A. Cook and Lila Fontes. Formal theories for linear algebra. In *24th International Workshop on Computer Science Logic*, pages 245–259, 2010.

[7] P. Hájek and P. Pudlák. *Metamathematics of First-order Arithmetic*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1993.

[8] Pavel Hrubeš 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.

[9] Pavel Hrubeš and Iddo Tzameret. Short proofs for the determinant identities. *SIAM J. Comput.*, 44(2):340–383, 2015. (A preliminary version appeared in Proceedings of the 44th Annual ACM Symposium on the Theory of Computing (STOC'12)).

[10] Emil Jeřábek. Dual weak pigeonhole principle, Boolean complexity, and derandomization. *Ann. Pure Appl. Logic*, 129(1-3):1–37, 2004.

[11] Jan Krajíček. *Bounded arithmetic, propositional logic, and complexity theory*, volume 60 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 1995.

[12] Gary L. Miller, Vijaya Ramachandran, and Erich Kaltofen. Efficient parallel evaluation of straight-line code and arithmetic circuits. *SIAM J. Comput.*, 17(4):687–695, 1988.

[13] Tonnian Pitassi and Iddo Tzameret. Algebraic proof complexity: Progress, frontiers and challenges. *ACM SIGLOG News*, 3(3), 2016.

[14] Michael Soltys. *The complexity of derivations of matrix identities*. PhD thesis, University of Toronto, Toronto, Canada, 2001.

[15] Michael Soltys and Stephen Cook. The proof complexity of linear algebra. *Ann. Pure Appl. Logic*, 130(1-3):277–323, 2004.

[16] Volker Strassen. Vermeidung von divisionen. *J. Reine Angew. Math.*, 264:182–202, 1973. (in German).

[17] Leslie G. Valiant, Sven Skyum, S. Berkowitz, and Charles Rackoff. Fast parallel computation of polynomials using few processors. *SIAM J. Comput.*, 12(4):641–644, 1983.