Polynomial Time Interactive Proofs for Linear Algebra with Exponential Matrix Dimensions and Scalars Given by Polynomial Time Circuits

In memory of Wen-tsun Wu (5/12/1919–5/7/2017)

Jean-Guillaume Dumas
U. Grenoble Alpes, CNRS
LJK, Grenoble, France

Erich L. Kaltofen
Dept. of Math., NCSU
Raleigh, NC, USA

Gilles Villard
CNRS,ENSL,Inria,UCBL
LIP, U. Lyon, France

Lihong Zhi
KLMM, AMSS, UCAS
Beijing, China

ABSTRACT

We present an interactive probabilistic proof protocol that certifies in $(\log N)^{O(1)}$ arithmetic and Boolean operations for the verifier the determinant, for example, of an $N \times N$ matrix over a field whose entries $a(i, j)$ are given by a single $(\log N)^{O(1)}$-depth arithmetic circuit, which contains $(\log N)^{O(1)}$ field constants and which is polynomial time uniform, for example, which has size $(\log N)^{O(1)}$. The prover can produce the interactive certificate within a $(\log N)^{O(1)}$ factor of the cost of computing the determinant. Our protocol is a version of the proofs for muggles protocol by Goldwasser, Kalai and Rothblum [STOC 2008, J. ACM 2015]. An application is the following: suppose in a system of k homogeneous polynomials of total degree $\leq d$ in the k variables $y_1, \ldots, y_k$ the coefficient of the term $y_1^{e_1} \cdots y_k^{e_k}$ in the i-th polynomial is the (hypergeometric) value $((i + e_1 + \cdots + e_k)!)/((i!)(e_1!) \cdots (e_k!))$, where $e!$ is the factorial of e. Then we have a probabilistic protocol that certifies (projective) solvability or inconsistency of such a system in $(k \log(d))^{O(1)}$ bit complexity for the verifier, that is, in polynomial time in the number of variables k and the logarithm of the total degree, $\log(d)$.

1. INTRODUCTION

Our interactive protocols for certifying the output of a symbolic computation are the nexus of four ideas: the first is the complexity class $\mathcal{NC}$ for uniform parallel circuits of polylogarithmic depth and the circuits that place linear algebra within the class. The second is the concept of representing symbolic computation input data by programs, among it Kaltofen's and Trager's [18] black box polynomials, rational functions and matrices. The third is the theory of probabilistically checkable proofs and Goldwasser's, Kalai's and Rothblum's [11, 12] (GKR) application to certifying that a high-cost computation which was delegated to a server (the prover) was actually performed, which is done by an interactive protocol with a verifier (GKR's muggle) who checks the output by an exponentially lower-cost computation, plus processing the input in linear time. The fourth is that classical polynomial algebra, which is the underlying mathematical method of solving systems of polynomial equations in k variables, can be reduced to linear algebra by coefficient vector shifts via multiplications by terms, for example, the Macaulay's matrices expressing the (multivariate) resultant.

ISSAC '17, July 25–28, 2017, Kaiserslautern, Germany

ACM ISBN 978-1-4503-5064-8/17/07…$15.00
https://doi.org/10.1145/3087604.3087640

Polynomial system solving has motivated our work: those resultant matrices have dimensions exponential in the number of variables k, and polynomial algebra in general has many algorithms which run in time exponential in k. The computations are often high-cost, which may be only doable by a server like a Google data center. The input coefficients are often not of exponential information complexity: the polynomial equations may be sparse in the number of terms, or the coefficients may be implicitly given by a formula, as in the example in the abstract above, or a program, say a determinant of a matrix. Of course, large matrices with computable entries occur outside polynomial algebra, see for instance [20].

The GKR interactive proof protocol is presented by Shafi Goldwasser, Yael Kalai and Guy Rothblum as a complexity result: computations on circuits of polylogarithmic depth with respect to the input size N whose directed acyclic graph (DAG) structure can be represented by log-space algorithms on the integers labelling the nodes are verified. Section 2 presents the GKR protocol in its entirety, but without a propabilistic analysis. The verifier scans the input once, in time $N^{1+o(1)}$, and checks poly-logarithmically many interactive steps in poly-logarithmically many bit operations, including flipping a public coin. The GKR protocol can be modified to arithmetic circuits [25], such as the determinant circuits by Kaltofen and Pan [17] (see Section 3). A saving is that arithmetic modulo a prime $p = N^{O(1)}$ does not need to be expanded to bit-by-bit operations in the protocol. Field arithmetic can be mixed with Boolean arithmetic for computing values from binary input data, such as $\binom{2(i+j-2)}{i+j-2}$, and for selecting data according to Boolean predicates, e.g., (2). We call circuits that perform Boolean arithmetic and operations on input field elements a ***hybrid*** arithmetic circuit.

Our verifier checks a computation on a family of circuits of size $N^{O(1)}$, or even $2^{(\log N)^{O(1)}}$, for $g_N(f_N(0), \ldots, f_N(N-1))$ in $(\log N)^{O(1)}$ bit communication and bit-operation complexity. Here g_N is a family of $(\log N)^{O(1)}$-depth circuits for a task such as the determinant, and f_N is a family of $(\log N)^{O(1)}$-depth circuits for the scalars such as the hypergeometric terms above; f_N can contain $(\log N)^{O(1)}$ input field constants. If the circuits f_N for the scalars are of size $(\log N)^{O(1)}$, they are input for the verifier. The circuit g_N and in the general case f_N are $N^{O(1)}$-sized and cannot be built by the verifier with poly-log complexity. The verifier rather accesses the circuits via algorithms that probe the circuit structures, which are called uniformity properties. The structures are input as $(\log N)^{O(1)}$ circuits of size $(\log N)^{O(1)}$ that determine the DAG structure of g_N and f_N. The circuit class $\mathcal{NC}$ used in GKR requires log-space algorithms, but we have relaxed the uniformity to polynomial-time and beyond (see Assumption 2.2). Because the input to the prover's circuit are the integers $0, \ldots, N-1$, the input scan in GKR is checkable by the verifier in $(\log N)^{O(1)}$ complexity (see (18)). Thus, the protocol we present here is distinguished from our previous work [9, 10, 16] whose communication and verifier complexity is at least

linear in N, that is, exponentially worse.

Our guiding example is the resultant matrix for polynomial systems, which has dimensions exponential in the number of variables k with entries being selected coefficients of the polynomials. There are still exponentially many (in k and the degrees) such coefficients. We assume the coefficients themselves are computed by circuits, for example, by tests on the degrees of the i-th equation that, for example, set all but polynomially many to zero, that is, represent a sparse system. We emphasize that sparsity is not a requirement for our polynomial-time certificates. In fact, the cost of verifying our certificate is polynomial in log(degree), and we could, in analogy to the notion of a supersparse polynomial [15], call our input system a ***superstructured*** system. An example is that we can certify in polynomial time that two univariate supersparse polynomials are relatively prime.

To achieve this complexity, with have some restrictions.

1. The verifier probes the prover's circuit structure using DAG structure test functions that for an integer node labelling scheme compute for each level ℓ if or if not a node of label α adds, or multiplies, etc., the values of the nodes β and γ on the previous level. Circuit complexity refers to such tests as a uniformity condition. Also the verifier's circuits for the scalars can be represented by DAG structure test functions instead of by their DAG. A multiplication tree for $N!$ for instance, has exponential size $O(N)$ but $O(\log N)$-sized simple DAG structure tests (cf. (11) and Remark 2.6 thereafter).
2. In order to obtain exponentially smaller proofs, we restrict the verifier to $(\log N)^{O(1)}$ complexity in rounds and bit operations. Using the GKR level-by-level certificates, we thus require all circuits to be of depth $(\log N)^{O(1)}$. Our weakest uniformity condition for both the circuit for the computed function and the circuit for the scalars is given in Assumption 2.2. It covers polynomial-time uniformity, that is that the DAG structure test functions are given by Boolean circuits of size $(\log N)^{O(1)}$. We recall that the node labels are integers of $(\log N)^{O(1)}$ bit length, as a bounded fan-in circuit of depth $(\log N)^{O(1)}$ has width $2^{(\log N)^{O(1)}}$.
3. Field elements, unlike the binary integers, cannot be tested to be equal to zero. The restriction is not universal: the prover and verifier have a simple protocol to certify that $N^{O(1)}$ elements are equal to zero (cf. Section 3 for the matrix rank). But the positions of those elements in the circuits must be testable, via their node labels, by the verifier. In other words, pivoting cannot be done, as those node labels depend on the input and would have to be communicated by the prover to the verifier, and are thus restricted in number to $(\log N)^{O(1)}$.
4. Without zero tests, divisions by field elements are excluded. Again, integer divisions in the Boolean part of the hybrid circuit are doable, for instance, by Newton iteration. The restriction again is not quite absolute. If all divisions are by non-zero field elements, we can modify our protocol to implement them (cf. Remark 2.5). For example, in processor-efficient parallel linear algebra, we have used randomization to avoid pivot testing [17]. Those random elements are chosen by the verifier or with a public coin. We restrict to $(\log N)^{O(1)}$ many random field elements that the verifier computes. We disallow $O(N)$, say, pseudo-random elements that the prover computes from a verifier provided $(\log N)^{O(1)}$-bit seed by a $(\log N)^{O(1)}$-depth circuit which the verifier confirms by the GKR protocol, since then the probability of correctness would depend on an additional hardness assumption.
5. The circuits can contain $(\log N)^{O(1)}$ field constants. To reach $(\log N)^{O(1)}$ binary verifier complexity, we think of the field of scalars as a finite field. Since $N \times N$ determinants of integer matrices can have $O(N(\log N))$ digits, we certify the determinant modulo a prime number p. We can test whether the determinant is 0 by choosing a random prime number with $(\log N)^{O(1)}$ digits. Also, on verifier input $i = N^{O(1)}$, the protocol can certify the i-th bit of the determinant in verifier cost $(\log N)^{O(1)}$. The prover cost would then be $N^{O(1)}$-fold higher, as the circuit is then purely binary.

Our prover complexity is $N^{O(1)}$, that is, exponential in the size of the certificate. The complexity includes the circuit for the algorithm, and the cost of evaluating the circuits for the scalars. In analogy to processor-efficient $\mathcal{NC}$ circuits, we can call the protocol ***prover-efficient*** if the certificate can be produced by the prover within a $(\log N)^{O(1)}$ factor of the width of the circuit for the algorithm. For the matrix determinant problem our protocol is prover-efficient (cf. Section 3). Restriction 4 has so far prevented us to have a prover-efficient matrix rank and the $\mathcal{NC}$ theory is still lacking of a processor-efficient circuit for the characteristic polynomial.

The soundness of the GKR protocol is based on the interaction between the exponential-time prover and the polynomial-time verifier. With cryptography, one can remove the interaction by a Fiat-Shamir heuristic, resulting in a static proof of polynomial system inconsistency, say, that can be checked in polynomial time in the number of variables and the logarithm of the degree. One cannot apply the protocol recursively to that check, exponentially reducing its size again, because the binary verification algorithm has polynomial depth [we thank Stephen M. Watt for the comment.]

2. CIRCUITS AS INPUTS

A problem in [11, 12], of which Thaler's scheme [25] is an algebraic implementation, is that the verifier cost is at least linear in the number of input bits. A polynomial-time verification hence seems to be ruled out for handling matrices with exponential dimensions such as resultant matrices. Here is an approach to shrink the input size for low complexity input scalars.

Our input scalars are in a finite field F, say $\mathsf{F} = \mathbb{Z}_p$ which denote the integers modulo a prime number p, and our algebraic circuit arithmetic for purpose of high probability of soundness of the verifier check will be done in a finite algebraic extension $\mathsf{K} \supseteq \mathsf{F}$. We assume that 2^n field elements $f_n(i) \in \mathsf{K}$ for $0 \le i \le 2^n - 1$ are represented by the hybrid extension circuit in Figure 1. The circuit can contain "passthru" nodes, which could have been emulated by arithmetic nodes as $v = v + w - w$ for any node w on the previous level. The circuit on inputs $\iota_1, \ldots, \iota_n \in \{0, 1\} \subseteq \mathsf{F}$ computes for the index integer $i = \iota_1 + 2\iota_2 + \cdots + 2^{n-1}\iota_n$, by interpreting the field elements $0, 1$ as the integers $0, 1$, the field element

$$\tilde{f}_n(i) \stackrel{\text{def}}{=} f_n(i) \stackrel{\text{def}}{=} \tilde{f}_n(\iota_1, \ldots, \iota_n) \in \mathsf{F}. \tag{1}$$

By $\widetilde{\ }$ we indicate an algebraic extension function and circuit that can be evaluated at both a bit vector and a vector of field elements. In the definition (1) we have overloaded the $\widetilde{\ }$'ed algebraic extension functions by writing single integer arguments in addition to vectors of field elements with the interpretation that the integers are converted to vectors of bits. The verification protocols will evaluate the extension circuits on random field elements $x_\nu \leftarrow \xi_\nu \in S \subseteq \mathsf{K}$ from a sufficiently large subset S. In summary, the $f_n(i)$ are the exponentially many input scalars that the single algebraic extensions circuit $\widetilde{C}_n$ represents via its algebraic extension function $\tilde{f}_n$.

The input to our computational problems is a hybrid extension circuit $\widetilde{C}_n$. The circuit is represented as a directed acyclic graph (DAG) in which each vertex is labelled by an integer and given an attribute on its functionality, such as "multiplication node". Here there is a Boolean part that can perform integer operations on $0/1$

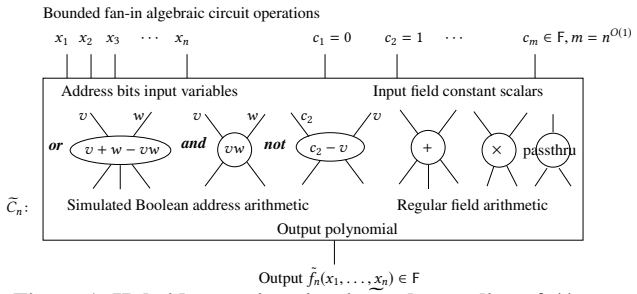

Figure 1: Hybrid extension circuit $\widetilde{C}_n$ that realizes $f_n(i)$

inputs, such as computing $\binom{i}{5}$ which for $i < 5$ is $= 0$, by simulating the Boolean operations on the field values 0 and 1. The output can then be a selected bit of $\binom{i}{5}$, or $\binom{i}{5}$ converted to a field element using the radix $2 = c_2 + c_2$. There is also an algebraic part that can supply actual input field elements to the circuit. The first two constants are the field elements 0 and 1. The zero element of the field can be synthesized inside the circuit as $x_1 - x_1$ by a binary subtraction, but field element 1 cannot, which is needed for wiring integer constants into the circuit. Polynomially in n many additional field elements can be supplied as input. Those elements could, for example, be output by the circuit for the inputs for certain values of i by a synthesized Boolean comparator and an arithmetic sum that is weighted by the Boolean predicates, for instance

$$(\,(i > q_1)\ \textbf{\textit{and}}\ (i \le q_2)\,)\ c_3 + (i \le q_1)\ c_4. \tag{2}$$

Here the comparisons are Boolean comparator circuits simulated by the field operations given in Figure 1, feeding the bits of the constants q_1 and q_2 using $c_1 = 0$ and $c_2 = 1$ as the second argument. Note that the input circuit can have unbounded fan-out. For arguments $v, w \in \{0, 1\} \subseteq \mathsf{F}$, the following field operations simulate the Boolean functions:

and	***or***	***not***	***xor***	***equiv***	***impl***
vw	$v + w$	$1 - v$	$v + w$	$1 + 2vw$	$1 - v$
	$-vw$		$-2vw$	$-v - w$	$+vw$

(3)

The hybrid sum of field values weighted by Boolean values (2) is a main ingredient in the GKR protocol, but we use it for representing structure and sparsity of our input matrices: in the above polynomial systems the input could be polynomially many (in the number of variables k) non-zero coefficients, that is a sparse polynomial system. We would have $n = \lceil \log(\binom{d+k-1}{k-1}) \rceil$ and the c_μ in $\widetilde{C}_n$ in Figure 1 for $\tilde{f}$ in (22) would hold those coefficients, which would be individually selected according to their equation index and term exponents (cf. (2)). Because the circuits are $n^{O(1)}$-size, they need not be uniform and can be the input. Note that the Boolean comparators compute a value in K for field inputs $x_\nu \leftarrow \xi_\nu \in \mathsf{K}$, which is the basis of the sum-check protocol [21].

REMARK 2.1. **Discussion of input circuit model (Part 1):** Our input circuits can have bounded fan-in (≤ 2) but unbounded fan-out. We suppose that the circuit is layered, that is each arithmetic node draws inputs from the layer directly above. In order to access values in earlier layers we therefore use passthru nodes with a single input. The node for the Boolean ***or*** thus occupies 2 layers, although one could introduce "super"nodes with 2 inputs and a fixed sized circuit with a single output in the GKR protocol. We will require the layering restriction when processing the GKR iteration over possibly exponentially many copies of our input circuits. All our circuits are hybrid arithmetic circuits and not pure Boolean circuits as in GKR. The GKR protocol can then be improved to fewer rounds of interaction as the circuits have smaller size. A limitation in the arithmetic version of GKR is that one cannot incorporate field element equality tests in the circuits and thus has to exclude divisions, all of which would be possible in the Boolean model. One can, of course, choose $\mathsf{F} = \mathbb{Z}_2$, in which case the field arithmetic is Boolean arithmetic, and the arithmetic version specializes to GKR's Boolean protocol. We note that if without checking the divisor the circuit on a given input does not divide by zero, divisions by field elements can be incorporated in the protocol; see Remark 2.5. □

An example of such a circuit would be the binomial coefficient $\binom{i}{5}$, which is $= 0$ for $i < 5$. Note that one has integer Boolean division circuits of depth $O(\log(i))$ [1].

Our version of GKR is an interactive proof to certify the value of a **exponential sized** formula or circuit with $f_n(i)$ inputs; note that there are 2^n indices i. We begin with the simplest but most important building block, the above mentioned sum-check protocol [21] for

$$\sigma_n = \textstyle\sum_{i=0}^{2^n-1} \tilde{f}_n(i). \tag{4}$$

To use sumcheck protocol of Lund et al., we must restrict our input circuit $\widetilde{C}_n$ in Figure 1 to have depth $O(\log(n))$; $\widetilde{C}_n$ no longer needs to be layered. The logarithmic depth restriction has two consequences:

1. Because in $\widetilde{C}_n$ each layer has at most twice as many nodes as the layer below, circuit depth $O(\log(n))$ implies circuit size $n^{O(1)}$. The verifier can evaluate $\tilde{f}_n$ with $n^{O(1)}$ field operations.
2. The total degree in $x_1, \ldots, x_n$ for the polynomial computed in each node at most doubles in the layer immediate below, so $\deg(\tilde{f}_n) = n^{O(1)}$.

Now we can perform the sum-check protocol for σ_n in (4) for the function $\tilde{f}_n$. In the sum-check interactive proof, the prover sends the verifier the polynomials

$$g_\mu(x_\mu) \stackrel{\text{def}}{=} \sum_{\iota_\nu \in \{0,\,1\},\ \mu+1 \le \nu \le n} \tilde{f}_n(r_1, \ldots, r_{\mu-1}, x_\mu, \iota_{\mu+1}, \ldots, \iota_n), \tag{5}$$

for $1 \le \mu \le n$, and the verifier sends inter-leaved uniformly selected random elements $r_1, r_2, \ldots, r_{n-1} \in S$ from a finite subset $S \subseteq \mathsf{K}$. The exchange is: prover: $g_1(x_1) \longrightarrow$ verifier, verifier: $r_1 \longrightarrow$ prover, prover: $g_2(x_2) \longrightarrow$ verifier, verifier: $r_2 \longrightarrow$ prover, ..., verifier: $r_{n-1} \longrightarrow$ prover, prover: $g_n(x_n) \longrightarrow$ verifier. The soundness check by the verifier is 1. for all μ with $2 \le \mu \le n$ check $g_\mu(0) + g_\mu(1) = g_{\mu-1}(r_{\mu-1})$; 2. for a uniformly selected random $r_n \in S$ check $g_n(r_n) = \tilde{f}_n(r_1, \ldots, r_n)$. If all checks succeed, then with high probability $g_1(0)+g_1(1) = \sum_{\iota_\nu \in \{0,1\},\ 1 \le \nu \le n} \tilde{f}_n(\iota_1, \ldots, \iota_n) = \sum_{i=0}^{2^n-1} \tilde{f}_n(i)$. Since $\deg(g_\mu) = n^{O(1)}$ by Consequence 2 and the input circuit $\widetilde{C}_n$ has $n^{O(1)}$ arithmetic gates by Consequence 1, the verifier runs in $n^{O(1)}$ field operations in K. For $\mathsf{K} = \mathbb{Z}_p$ with $\log(p) = n^{O(1)}$ the verifier uses $n^{O(1)}$ bit operations to verify the summation σ_n with 2^n terms in (4).

Next, we demonstrate GKR on a product-check protocol for

$$\tau_n = \textstyle\prod_{i=0}^{2^n-1} \tilde{f}_n(i). \tag{6}$$

At the same time, ***we shall remove the logarithmic depth restriction on*** $\widetilde{C}_n$. Our example encompasses all of the GKR protocol and all uniformity conditions on the arising circuits. The value τ_n is computed by the exponential-sized hybrid extension circuit $\widetilde{P}_n$ of Figure 2. The bottom part of the $\widetilde{P}_n$ is a perfect binary tree with multiplications in its nodes. The DAG corresponding to the tree has a highly uniform structure: see (11) below. On the leaves of the binary tree are connected 2^n identical copies of the hybrid input extension circuit $\widetilde{C}_n$. Each of the $n + m$ inputs is connected to $2^n(n + m)$ identical copies of hybrid input "decoder" circuits $\widetilde{D}_n$. The detailed input lines are shown in the bottom left inset of Figure 2. Here $b_{m+n} = \lceil \log_2(n + m) \rceil$ is the minimum number of bits required to represent κ in the range $0 \le \kappa \le n + m - 1$. The idea is that the decoder selects the appropriate input values as output whose positions are given in the κ input portion of the decoder, which is then appropriately set on input. The decoder also can select an input

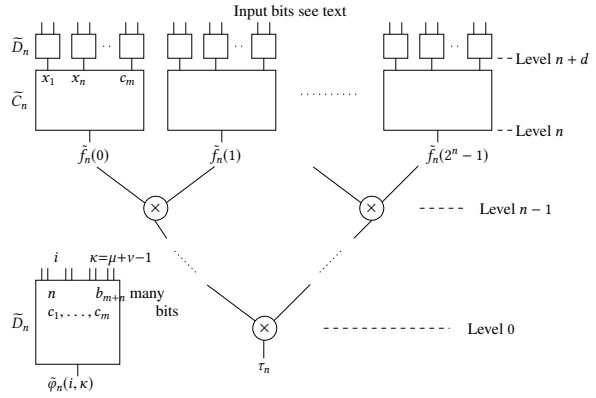

Figure 2: Hybrid extension circuit $\widetilde{P}_n$ that realizes $\tilde{\tau}_n$

constant c_μ, which is a node of that value inside $\widetilde{D}_n$ with fan-in zero. Note that the circuit $\widetilde{D}_n$ is no longer layered in the way that each node receives data from the immediate previous layer as was the case for $\widetilde{C}_n$ and is the case for the binary multiplication tree. We give a hybrid Boolean-arithmetic sum (cf. (2)) for the algebraic extension function $\widetilde{\varphi}_n$:

$$\widetilde{\varphi}_n(i,\kappa) = \Big(\textstyle\sum_{\nu=1}^{n}(\kappa < n \textbf{ \textit{and} } \kappa+1=\nu)\ \iota_\nu\Big) + \Big(\textstyle\sum_{\mu=1}^{m}(\kappa \geq n \textbf{ \textit{and} } \kappa-n+1=\mu)\ c_\mu\Big). \tag{7}$$

In (7) ι_ν is the ν-th bit of i. In the extension function, the bits of i and κ will be set to field elements with ι_ν the corresponding input element, which the comparator circuits, Boolean circuits and binary arithmetic circuits process "blindly" according to circuit structure. Note that

$$\widetilde{\varphi}_n(i,\kappa) \stackrel{\text{def}}{=} \begin{cases} \iota_{\kappa+1} & \text{for } 0 \leq \kappa \leq n-1, \\ c_{\kappa-n+1} & \text{for } n \leq \kappa \leq n+m-1, \\ 0 & \text{for } \kappa \geq n+m. \end{cases} \tag{8}$$

Now we can set the inputs to our computation. The input to the $(\kappa+1)$'st circuit $\widetilde{D}_n$ of the $(i+1)$'st copy of $\widetilde{C}_n$ is the pair (i,κ) represented by the $n+b_{m+n}$ input bits, with $b_{n+m} = \lceil \log_2(n+m) \rceil$. Note that only the prover will actually use those inputs.

The reason for the hybrid address decoder circuits $\widetilde{D}_n$ above the $\widetilde{C}_n$ is that they can be implemented by (7) with depth $O(\log(n))$ and therefore can be used in a sum-check protocol. Integer addition, subtraction, and comparison is done by parallel prefix look-ahead circuits [2]. We will need the algebraic extension circuit for equality testing below and give the formula now. The test has as input two bit vectors $y_1,\ldots,y_b$ and $\iota_1,\ldots,\iota_b$ and returns 1 if each pair of bits in the vectors are equivalent, else it returns 0:

$$\begin{aligned} \tilde{\text{eq}}(y,i) &\stackrel{\text{def}}{=} \textstyle\prod_{\theta=1}^{b}(y_\theta \textbf{ \textit{equiv} } \iota_\theta) \\ &= \textstyle\prod_{\theta=1}^{b}(1+2\iota_\theta y_\theta - \iota_\theta - y_\theta), \end{aligned} \tag{9}$$

where $\deg_{y_1,\ldots,y_b}(\tilde{\text{eq}}) = b$. Below in, for example, (11), as above in (7), we also write $(y=i)$ for $\tilde{\text{eq}}(y,i)$.

REMARK 2.2. One does not need the decoder circuits $\widetilde{D}_n$ if the circuits $\widetilde{C}_n$ have depth $O(\log(n))$, in which case one may then use $\widetilde{C}_n$ directly. The verifier then evaluates the hybrid extension circuits $\widetilde{C}_n$ at a vector of random field elements, and the only uniformity condition required is that the verifier can perform the evaluation in $n^{O(1)}$ arithmetic steps (cf. Consequence 1 above). Therefore, the logarithmic-depth integer division circuits and multiple integer product circuits in [1] can be incorporated into $\widetilde{C}_n$. However, our version of the GKR protocol will place the constant B for $\text{depth}(\widetilde{C}_n) \leq B\log(n)$ in the exponent of the verifier cost, because $\deg(\tilde{f}_n) \leq 2^{\text{depth}(\widetilde{C}_n)} \leq n^B$. With decoder circuits $\widetilde{D}_n$, the depth of $\widetilde{C}_n$ is only restricted to $n^{O(1)}$ and $\widetilde{C}_n$ can even be of exponential size with the uniformity in structure (see Remark 2.6). □

We now assume that at level ℓ of $\widetilde{P}_n$ in Figure 2 the nodes are labeled $\alpha = 0,1,\ldots,L_\ell - 1$, where α is represented as an s_ℓ-bit integer with $s_\ell = \lceil \log_2(L_\ell) \rceil$. Note that $s_\ell = \ell$ for $\ell = 0,\ldots,n$ in the levels of the perfect binary tree in $\widetilde{P}_n$ of Figure 2. At level $n+d$ we have $s_{n+d} = n + b_{m+n}$, which is the number of bits on the labels to all $(n+m)2^n$ inputs of the 2^n copies of $\widetilde{C}_n$ in $\widetilde{P}_n$. The heart of the GKR protocol is the recursive property of the algebraic extension function $\widetilde{V}_\ell(z_1,\ldots,z_{s_\ell})$ for each level ℓ in (13) below, where $\widetilde{V}_\ell(z_1,\ldots,z_{s_\ell})$ is defined as

$$\widetilde{V}_\ell(z_1,\ldots,z_{s_\ell}) \stackrel{\text{def}}{=} \textstyle\sum_{\alpha=0}^{2^{s_\ell}-1} \tilde{\text{eq}}u(z_1,\ldots,z_{s_\ell},\alpha)\ W_{\ell,\alpha}, \tag{10}$$

where $W_{\ell,\alpha}$ is the field element value of node α at level ℓ when evaluating the circuit on the given inputs, and $W_{\ell,\alpha} = 0$ for $\alpha \geq L_\ell$, that is when the node label α is out of range. We thus have $\widetilde{V}_\ell(\alpha) = W_{\ell,\alpha}$; note in both $\tilde{\text{eq}}u(z_1,\ldots,z_{s_\ell},\alpha)$ and $\widetilde{V}_\ell(\alpha)$ the overloaded usage of α for the vectors of bits in α. We will discuss the node labeling further in the Remark 2.4 below. Furthermore, the prover and verifier in the protocol must be given algebraic extension circuits $\tilde{\text{add}}_\ell$, $\tilde{\text{mul}}_\ell$ and $\tilde{\text{thru}}_\ell$, and preferably also have circuits for $\tilde{\text{sub}}_\ell$, $\tilde{\text{or}}_\ell$, $\tilde{\text{not}}_\ell$, $\tilde{\text{xor}}_\ell$, $\tilde{\text{equiv}}_\ell$ and $\tilde{\text{impl}}_\ell$, that implement the following *DAG structure test functions* on their binary inputs representing the bits of a node label α on level ℓ and two node labels β and γ on level $\ell+1$:

1. $\tilde{\text{add}}_\ell(\alpha,\beta,\gamma)$ is 1 if node α on level ℓ is an addition node with nodes β and γ as inputs, and is 0 otherwise; $\tilde{\text{sub}}_\ell$ and $\tilde{\text{mul}}_\ell$ (for multiplication and Boolean ***and***) are defined accordingly.
2. $\tilde{\text{or}}_\ell(\alpha,\beta,\gamma)$ is 1 if node α on level ℓ is a Boolean ***or*** "super"node with nodes β and γ as inputs, and is 0 otherwise; $\tilde{\text{xor}}_\ell$, $\tilde{\text{equiv}}_\ell$, and $\tilde{\text{impl}}_\ell$ are defined accordingly. By not emulating those nodes by their arithmetic equivalents (3) the functions should be easier to implement on the Boolean parts of the hybrid circuits.
3. $\tilde{\text{not}}_\ell(\alpha,\beta,\gamma)$ is 1 if node α on level ℓ is a ***not*** node with node β as input and $\gamma = \beta$, and is 0 otherwise; $\tilde{\text{thru}}_\ell$ is defined accordingly for passthru. We do not emulate the Boolean ***not*** by a subtraction and do not require the field constant 1 via pass thru.

We require the following property.

ASSUMPTION 2.1. ***Log-depth.*** *All algebraic extension circuits for* $\tilde{\text{add}}_\ell$, $\tilde{\text{sub}}_\ell$, $\tilde{\text{mul}}_\ell$, …, $\tilde{\text{thru}}_\ell$ *are for all levels* ℓ *of depth* $O(\log(n))$, *hence of degree and size* $n^{O(1)}$, *which the verifier is able to evaluate at field elements as inputs in polynomial time in* n.

For example, in $\widetilde{P}_n$ for levels $\ell = 0,\ldots,n-1$ we have $\tilde{\text{add}}_\ell$, $\tilde{\text{sub}}_\ell$ and $\tilde{\text{thru}}_\ell$ always return 0 and

$$\tilde{\text{mul}}_\ell(\alpha,\beta,\gamma) = (\beta = 2\alpha \textbf{ \textit{and} } \gamma = 2\alpha+1). \tag{11}$$

REMARK 2.3. If the circuits $\widetilde{C}_n$ are of size $n^{O(1)}$ represented by their DAGs, the DAG structure functions can be built by the prover and the verifier as a tree of comparisons, e.g.,

$$\tilde{\text{add}}_\ell(u,v,w) = \sum_{\bar{\alpha},\bar{\beta},\bar{\gamma}} \tilde{\text{eq}}u(\bar{u},\bar{v},\bar{w},\bar{\alpha},\bar{\beta},\bar{\gamma})\ \tilde{\text{add}}_\ell(\bar{\alpha},\bar{\beta},\bar{\gamma}). \tag{12}$$

Here the $\bar{u},\ldots\bar{\gamma}$ refers to the portions of the inputs bits of the node labels that lie within $\widetilde{C}_n$ (see also Remark 2.4 below) and the bits representing i in the i-th copy of $\widetilde{C}_n$ are not tested. The verifier can compute the vector of values $\tilde{\text{add}}_\ell(\bar{\alpha},\bar{\beta},\bar{\gamma})$ by traversing the DAG for $\widetilde{C}_n$ and evaluate (12) in $n^{O(1)}$ field operations. It is here that the input becomes logarithmic in the size of $\widetilde{P}_n$. □

Finally, we can express $\widetilde{V}_\ell(z_1,\ldots,z_{s_\ell}) \in \mathsf{K}[z_1,\ldots,z_{s_\ell}]$ recursively in terms of $\widetilde{V}_{\ell+1}$ as a sum of the form (4):

$$\widetilde{V}_\ell(z^{[\ell]}) = \sum_{\alpha=0}^{2^{s_\ell}-1}\ \sum_{\beta=0}^{2^{s_{\ell+1}}-1}\ \sum_{\gamma=0}^{2^{s_{\ell+1}}-1} \tilde{\text{eq}}u(z^{[\ell]},\alpha)\widetilde{F}_\ell(\alpha,\beta,\gamma), \tag{13}$$

where $z^{[\ell]} = [z_1,\ldots,z_{s_\ell}]$, and (compare with (3))

$$\begin{aligned} \widetilde{F}_\ell(\alpha,\beta,\gamma) \stackrel{\text{def}}{=}\ & \tilde{\text{add}}_\ell(\alpha,\beta,\gamma)\,(\widetilde{V}_{\ell+1}(\beta)+\widetilde{V}_{\ell+1}(\gamma)) \\ & + \tilde{\text{mul}}_\ell(\alpha,\beta,\gamma)\,\widetilde{V}_{\ell+1}(\beta)\,\widetilde{V}_{\ell+1}(\gamma) \end{aligned}$$

$$+ \tilde{\text{or}}_\ell(\alpha,\beta,\gamma)\left(\widetilde{V}_{\ell+1}(\beta)+\widetilde{V}_{\ell+1}(\gamma)-\widetilde{V}_{\ell+1}(\beta)\,\widetilde{V}_{\ell+1}(\gamma)\right)$$
$$+ \ldots \text{ [terms for } \tilde{\text{sub}}_\ell, \tilde{\text{xor}}_\ell, \tilde{\text{equiv}}_\ell, \tilde{\text{impl}}_\ell, \text{ cf. (3)]}$$
$$+ \tilde{\text{not}}_\ell(\alpha,\beta,\gamma)\,\left(1-\widetilde{V}_{\ell+1}(\beta)\right) + \tilde{\text{thru}}_\ell(\alpha,\beta,\gamma)\,\widetilde{V}_{\ell+1}(\beta). \quad (14)$$

Note that the constants in (14) are not pass thru constants. The fact that the triple sum in (13) for $\widetilde{V}_\ell(z^{[\ell]})$ equals its definition (10) follows easily: for a node label α on level ℓ we have

$$W_{\ell,\alpha} = \sum_{\beta=0}^{2^{s_{\ell+1}}-1}\sum_{\gamma=0}^{2^{s_{\ell+1}}-1}\widetilde{F}_\ell(\alpha,\beta,\gamma), \quad (15)$$

because by the definition (10) for level $\ell+1$ we have $\widetilde{V}_{\ell+1}(\beta) = W_{\ell+1,\beta}$ and $\widetilde{V}_{\ell+1}(\gamma) = W_{\ell+1,\gamma}$. If the node label α is out of range the rights side of (15) returns the field element of 0, as is required, since all DAG structure test functions return 0. We give the formula corresponding to the definition of $\widetilde{V}_\ell(z^{[\ell]})$ in (10) for the top level $n+d$ in (18) below. On level $n+d$ we have that $\widetilde{V}_{n+d}(\alpha) = 0$ for node labels out of range, namely for $\alpha \geq L_{n+d} = (n+m)2^n$, from the third case of (8).

The prover and verifier perform an interactive protocol recursively for $\ell = 0, 1, \ldots, n+d$ where at each level ℓ, for a then given s_ℓ-dimensional vector of random field values $\vec{r}^{\,[\ell]} \in S^{s_\ell}$, they perform a sum-check protocol for $\widetilde{V}_\ell(\vec{r}^{\,[\ell]})$ defined by (13). By the definition (10) for level $\ell+1$ and (9), $\deg_{z^{[\ell+1]}}(\widetilde{V}_{\ell+1}(z^{[\ell+1]})) \leq s_{\ell+1}$, where $s_{\ell+1}$ is the number of bits in the node labels at level $\ell+1$ and the "$\leq$" accounts for the fact that all nodes at level $\ell+1$ could evaluate to 0 on input. If for all ℓ we have $s_\ell = n^{O(1)}$, which is certainly the case if $\text{size}(\widetilde{C}_n) = n^{O(1)}$, then we have $\deg_{u,v,w}\left(\,\tilde{\text{equ}}(\vec{r}^{\,[\ell]},u)\,\widetilde{F}_\ell(u,v,w)\,\right) = n^{O(1)}$, because by the log-depth Assumption 2.1 the degrees of the algebraic extensions circuits for $\tilde{\text{add}}_\ell$, $\tilde{\text{sub}}_\ell$, $\tilde{\text{mul}}_\ell$, ..., $\tilde{\text{thru}}_\ell$ are thus bounded. The prover and verifier exchange g_μ of (5) and random field elements, until the verifier completes the sum-check of $\widetilde{V}_\ell(\vec{r}^{\,[\ell]})$ by testing

$$g_{s_\ell+2s_{\ell+1}}(\rho^{[3]}_{s_{\ell+1}}) \stackrel{?}{=} \tilde{\text{equ}}(r^{[\ell]},\vec{\rho}^{\,[1]})\,\widetilde{F}_\ell(\vec{\rho}^{\,[1]},\vec{\rho}^{\,[2]},\vec{\rho}^{\,[3]}), \quad (16)$$

with $\vec{\rho}^{\,[1]} \in S^{s_\ell}$, $\vec{\rho}^{\,[2]} \in S^{s_{\ell+1}}$ and $\vec{\rho}^{\,[3]} \in S^{s_{\ell+1}}$ verifier chosen random vectors whose entries were communicated one after another, the last one being $\rho^{[3]}_{s_{\ell+1}}$. By (9) the verifier can compute $\tilde{\text{equ}}(r^{[\ell]},\vec{\rho}^{\,[1]})$ in polynomial time in n, as well as, by the Assumption 2.1, $\tilde{\text{add}}_\ell(\vec{\rho}^{\,[1]},\vec{\rho}^{\,[2]},\vec{\rho}^{\,[3]})$, ..., $\tilde{\text{thru}}_\ell(\vec{\rho}^{\,[1]},\vec{\rho}^{\,[2]},\vec{\rho}^{\,[3]})$.

The verifier, however, cannot compute $\widetilde{V}_{\ell+1}(\vec{\rho}^{\,[2]})$ and $\widetilde{V}_{\ell+1}(\vec{\rho}^{\,[3]})$ in polynomial time in n, as that requires all values of the nodes on level $\ell+1$. To verify those two values, the protocol ingeniously recurses to level $\ell+1$. A clever ("standard" by [12]) subprotocol is used to reduce the check of 2 values of $\widetilde{V}_{\ell+1}$ to a single value [25, Remark 2]. The prover first communicates the univariate polynomial in y,

$$\psi_\ell(y) = \widetilde{V}_{\ell+1}(\,(1-y)\vec{\rho}^{\,[2]} + y\vec{\rho}^{\,[3]}) \in \mathsf{K}[y], \quad (17)$$

where $\deg(\psi_\ell) \leq s_{\ell+1}$, after which the verifier communicates a random field element $r^* \in S$ and the protocol recurses to level $\ell+1$ to certify $\widetilde{V}_{\ell+1}(\vec{r}^{\,[\ell+1]})$ with $\vec{r}^{\,[\ell+1]} = (1-r^*)\vec{\rho}^{\,[2]} + r^*\vec{\rho}^{\,[3]}$. Finally, the verifier compares the certified value with $\psi_\ell(r^*)$, and uses $\psi_\ell(0) = \widetilde{V}_{\ell+1}(\vec{\rho}^{\,[2]})$ and $\psi_\ell(1) = \widetilde{V}_{\ell+1}(\vec{\rho}^{\,[3]})$ for completing the computation of $\widetilde{F}_\ell(\vec{\rho}^{\,[1]},\vec{\rho}^{\,[2]},\vec{\rho}^{\,[3]})$ for the final step (16).

The recursion for verifying $\widetilde{P}_n$ stops at level $n+d$, where d is the depth of $\widetilde{C}_n$. At that level we define

$$\widetilde{V}_{n+d}(z^{[n+d]}) \stackrel{\text{def}}{=} \sum_{i=0}^{2^n-1}\sum_{\kappa=0}^{2^{b_{n+m}}-1} \tilde{\text{equ}}(z^{[n+d]},i,\kappa)\widetilde{\varphi}_n(i,\kappa), \quad (18)$$

where $z^{[n+d]} = [z_1,\ldots,z_{n+b_{n+m}}]$. Note that the integers $i + 2^n\kappa$ are assumed to be the labels α on level $n+d$ of the output nodes in $\widetilde{D}_n$ that are the inputs to the copies of $\widetilde{C}_n$ in $\widetilde{P}_n$ of Figure 2. The κ part of the bit representation corresponds to $\bar{\beta}$ and $\bar{\gamma}$ in (12) of Remark 2.3. By (7) and (9) we have $\deg_{x,y}(\tilde{\text{equ}}(z^{[n+d]},x,y) \times \widetilde{\varphi}_n(x,y)) = n^{O(1)}$. The verifier can in polynomial time in n evaluate $\tilde{\text{equ}}(\vec{r}^{\,[n+d]},\vec{\rho}^{\,[1]},\vec{\rho}^{\,[2]})\,\widetilde{\varphi}_n(\vec{\rho}^{\,[1]},\vec{\rho}^{\,[2]})$ at vectors of random field elements $\vec{r}^{\,[n+d]} \in S^{n+b_{n+m}}$, $\vec{\rho}^{\,[1]} \in S^n$ and $\vec{\rho}^{\,[2]} \in S^{b_{n+m}}$. A sum-check protocol for $\widetilde{V}_{n+d}(\vec{r}^{\,[n+d]})$ stops the recursion.

The recursion begins at level 0. There is a single node, so $\widetilde{V}_0$ has no argument, and is in $\widetilde{P}_n$ equal to $\widetilde{V}_0 = \widetilde{V}_1(0)\widetilde{V}_1(1)$. In the case of $\widetilde{P}_n$ in Figure 2 we have $z^{[1]} = z_1$ and $\widetilde{V}_1(z_1) = \sum_{\alpha=0}^{1}\tilde{\text{equ}}(\alpha,z_1)\,\widetilde{V}_2(2\alpha)\widetilde{V}_2(2\alpha+1)$, having evaluated the DAG structure test functions (11) explicitly for (14,13). The recursion begins with the prover communicating the (at most linear) polynomial $\psi_0(y) = \widetilde{V}_1(y)$ and the verifier choosing a random $r^* \in S$ and asking for a proof of $\widetilde{V}_1(r^*)$ by recursion. Finally, if the comparison with $\psi_0(r^*)$ succeeds, the verifier computes $\tau_n = \psi_0(0)\psi_0(1) = \widetilde{V}_1(0)\widetilde{V}_1(1)$.

REMARK 2.4. **Node labels.** We have used the perfect tree for computing the product (6) so that the DAG structure test functions $\tilde{\text{add}}_\ell$, etc., can be easily described; see (11) and Remark 2.3. For the product tree, one could incorporate (11) directly in the definition of $\widetilde{V}_\ell$: see [25, Section 5.3.1]. In Section 3 we will place determinant circuits below the copies of input circuits $\widetilde{C}_n$. For those circuits, we wish to structure the bits in the labels of the arithmetic nodes on level ℓ according to their role in the algorithm. We can encode that role in blocks of bits of the node label α on level ℓ, and not have all bit patterns correspond to actual nodes. The reason was already stated above: nodes corresponding to a label α out of range have $\widetilde{V}_\ell(\alpha) = 0$. For example, we could have labeled the nodes on level $n+d$ with integer triples (i,ν,μ) instead of the pairs (i,κ) and used in place of (7) the hybrid algebraic extension circuit for

$$\widetilde{\varphi}_n(i,\nu,\mu) = \left(\sum_{\theta=1}^{n}(\mu = 0 \textbf{ and } \theta = \nu)\;\iota_\nu\right) + \left(\sum_{\theta=1}^{m}(\nu = 0 \textbf{ and } \theta = \mu)\;c_\mu\right). \quad (19)$$

Each node label on level $n+d$ then has either $\mu = 0$ or $\nu = 0$, and we have $\varphi_n(i,\nu,\mu) = 0$ for any invalid node label with both $\nu \neq 0$ and $\mu \neq 0$, by (19). The circuit for (19) has a smaller depth than the circuit for (7), hence the polynomials in the sum-check protocol have smaller degrees. However, there are more bits in the representation of (i,ν,μ) than of (i,κ), hence the sum-check protocol would then have more rounds. □

REMARK 2.5. **Division nodes.** If the verifier knows that no divisions by zero occur, which may be accomplished by inputting random elements among the c_μ, the protocol can be amended. One can push the divisions to the very last arithmetic node, by simulating rational number arithmetic in each node. Every field element $c \in \mathsf{K}$ in the circuit is represented by a pair of elements (c',c''), a formal numerator and denominator, with $c'' \neq 0$ and $c = c'/c''$. We implement the arithmetic operations as unreduced rational number operations: $c_1 \pm c_2$ as $(c_1'c_2'' \pm c_2'c_1'', c_1''c_2'')$, c_1c_2 as $(c_1'c_2', c_1''c_2'')$ and c_1/c_2 as $(c_1'c_2'', c_2'c_1'')$. If there is no division by zero in the circuit, the c'' will be non-zero throughout the evaluation, which is proven by induction on the levels. The final node is the single division, $\psi_0(0)/\psi_0(1) = \widetilde{V}_1(0)/\widetilde{V}_1(1)$, where $\widetilde{V}_1(0)$ is the formal numerator and $\widetilde{V}_1(1)$ the formal denominator value of the output (see the paragraph immediately above Remark 2.4). By assumption of no zero divisions we have $\psi_0(0) \neq 0$, but $\psi_0(0) \neq 0$ does not guarantee that there is no earlier division by zero: dividing $(1,1)$ by $(0,1)$ and then $(1,1)$ by the resulting $(1,0)$ results in $(0,1)$. □

REMARK 2.6. **Discussion of input circuit model (Part 2).** In order to have verifier complexity $n^{O(1)}$ we require $d = \text{depth}(\widetilde{C}_n) = n^{O(1)}$. In Remark 2.3 we have shown how the DAG structure test functions $\tilde{\text{add}}_\ell$, etc., of depth $O(\log(n))$ can be built from a DAG

of polynomial size. In the GKR protocol, the verifier evaluates the DAG structure test functions, but not $\widetilde{C}_n$. Instead, the verifier runs the protocol up the levels of the DAG for $\widetilde{C}_n$ and then evaluates the decoder circuits $\widetilde{D}_n$. Therefore, the input circuits $\widetilde{C}_n$ can have exponential size and be represented by their $\tilde{\text{add}}_\ell$, $\tilde{\text{sub}}_\ell$, $\tilde{\text{mul}}_\ell$, …, $\tilde{\text{thru}}_\ell$ that must satisfy the log-depth Assumption 2.1.

An example would be a circuit $\widetilde{C}_n$ that computes $\tilde{f}_n(i) = \binom{2i}{i}$ for $i = 0, 1, \ldots, 2^n - 1$ in the field F, say $\mathsf{F} = \mathbb{Z}_p$ with p an $n^{O(1)}$-digit prime number, so that the field arithmetic is also polynomial time in n, and $p > 2^n$ so that there are no divisions by zero. In $\widetilde{C}_n$ we shall compute $\binom{2i}{i}$ as a hypergeometric sequence $\binom{2i}{i} = \frac{(2i-1)(2i)}{i^2}\binom{2(i-1)}{i-1}$. Our example thus demonstrates the circuit construction for all hypergeometric sequences. We now describe $\widetilde{C}_n$. We first initialize on a certain level 2^n blocks $\vec{\Phi}_j$ of n nodes ($0 \le j \le 2^n - 1$) that represent the values $0, 1, \ldots, j, \ldots, 2^n - 1$ as n-bit binary integers. We can achieve those values in $O(n \log(n))$ depth by a parallel prefix circuit for $\vec{\Phi}_j = \sum_{\delta=1}^{j} 1$ with $j \ge 1$, where 1 is represented as a binary n-bit integer using the inputs $c_1 = 0$ and $c_2 = 1$ as the bit values, and where the prefix operation is binary look-ahead addition of depth $O(\log(n))$ on n bit integers. Note that inputs for c_1 and c_2 now have fan-out exponential in n. The nodes $\vec{\Phi}_0$ are initialized by passing thru $c_1 = 0$. In the next layers we filter all $\vec{\Phi}_j$, whose value is $\vec{\Phi}_j = j$ in binary representation, into 2^n nodes $\Psi_j^{[\text{num}]}$ and $\Psi_j^{[\text{den}]}$ ($0 \le j \le 2^n - 1$) of field elements, containing

$$\Psi_j^{[\text{num}]} = (\vec{\Phi}_j = 0 \textbf{ or } \vec{\Phi}_j > i) + (\vec{\Phi}_j \ne 0 \textbf{ and } \vec{\Phi}_j \le i)(2\vec{\Phi}_j - 1)\, 2\vec{\Phi}_j,$$
$$\Psi_j^{[\text{den}]} = (\vec{\Phi}_j = 0 \textbf{ or } \vec{\Phi}_j > i) + (\vec{\Phi}_j \ne 0 \textbf{ and } \vec{\Phi}_j \le i)\vec{\Phi}_j^2.$$

Here i is an input to $\widetilde{C}_n$ as an n-bit integer, which also has been passed thru. In the above circuits for $\Psi_j^{[\text{num}]}, \Psi_j^{[\text{den}]}$ the Boolean tests are performed by binary integer arithmetic on the values represented by $\vec{\Phi}_j$ and i, yielding as output values a single field element of value 0 or 1. The values $(2\vec{\Phi}_j - 1)\, 2\vec{\Phi}_j$ and $\vec{\Phi}_j^2$ are computed by field arithmetic after converting the binary integer representations $\vec{\Phi}_j$ to field elements of that value from their 0/1 bit values ι_ν, namely $\sum_{\nu=1}^{n} \iota_\nu 2^{\nu-1}$. Finally, we compute $\binom{2i}{i} = (\prod_{j=0}^{2^n-1} \Psi_j^{[\text{num}]})/(\prod_{j=0}^{2^n-1} \Psi_j^{[\text{den}]})$ via two trees of height n. The last operation introduces a division, and the remaining circuitry can use the formal numerators and denominators introduced in Remark 2.5.

The single hybrid extension circuit $\widetilde{C}_n$ for $\binom{2i}{i}$, as an element in the field F, has width $O(n2^n)$ and depth $O(n \log(n))$. However, the circuit has a regular structure because of the regularity of parallel prefix, which is also used in look-ahead addition [2], where the ***xor*** "super"nodes are useful. The $O(n \log(n))$ DAG structure test functions $\tilde{\text{add}}_\ell$, $\tilde{\text{sub}}_\ell$, $\tilde{\text{mul}}_\ell$, …, $\tilde{\text{thru}}_\ell$, which manipulate the node labels of $O(n)$ bits, possibly by modular arithmetic using division circuits of $O(\log(n))$ depth [1] and integer constants with $O(n)$ bits, can satisfy the requirements of the log-depth Assumption 2.1. Note that the circuits need not be log-space uniform, only polynomial-time uniform: the prover and verifier evaluate them.

In summary, we have constructed a GKR protocol that can verify, for instance, $\sigma_{n,p} = \sum_{i=0}^{2^n-1} \binom{2i}{i} \bmod p$ for a prime number p with $\log(p) = n^{O(1)}$ in bit complexity $n^{O(1)}$. □

REMARK 2.7. **Discussion of input circuit model (Part 3).** We now show how to relax the log-depth Assumption 2.1 in our protocol. This will allow us to consider circuits with structure test functions of degree possibly exponential, whilst keeping a protocol with communication and verifier complexity $n^{O(1)}$. If the structure test degree is exponential then the polynomials g_μ of (5) in the corresponding sumchecks cannot be sent by the prover. We use an idea similar to the one used in [12, Section 4.2], where the evaluation of the structure test functions itself is delegated to the prover and proved using an interactive sub-protocol. Here we use a sub-protocol recursively that is not limited to the test functions. We address the evaluation of (13) globally by delegating it to the prover, and proving it thanks to a sub-protocol analogous to the one used for the tree of Figure 2.

The following uniformity condition is sufficient for a protocol of verifier complexity $n^{O(1)}$.

ASSUMPTION 2.2. ***Polynomial-depth.*** *All algebraic extension circuits for the DAG structure test functions* $\tilde{\text{add}}_\ell$, $\tilde{\text{sub}}_\ell$, $\tilde{\text{mul}}_\ell$, …, $\tilde{\text{thru}}_\ell$ *are for all levels* ℓ *of* ***depth*** $n^{O(1)}$ *and possibly exponential size (or even size* $2^{n^{O(1)}}$*), which are given by their own DAG structure test functions of depth* $O(\log(n))$*, hence degree and size* $n^{O(1)}$*, which the verifier is able to evaluate at field elements in* $n^{O(1)}$ *time. If some (or all) circuits for the DAG structure test functions* $\tilde{\text{add}}_\ell$*, etc., for the initial input circuit are given by DAGs of* ***size*** $n^{O(1)}$*, then the required DAG structure test circuits of depth* $O(\log n)$ *for those DAGs can be constructed in* $n^{O(1)}$ *time by the prover and verifier as in Remark 2.3.*

We now sketch how to modify our protocol to account for the weaker log-depth Assumption 2.2 for the DAG structure test function. Instead of performing a sumcheck protocol for $\widetilde{V}_\ell(\vec{r}^{\,[\ell]})$ defined by (13) and (14), the prover and verifier engage in a tree evaluation protocol for a tree analogous to Figure 2: the nodes in the large perfect binary tree are addition nodes, adding up the $2^{s_\ell + 2s_{\ell+1}}$ values $\tilde{\text{equ}}(\vec{r}^{\,[\ell]}, \alpha)\ \widetilde{F}_\ell(\alpha, \beta, \gamma)$. The function $\tilde{f}_{s_\ell+2s_{\ell+1}}$ has inputs $\alpha, \beta, \gamma, v_1, v_2, z^{[\ell]}$ and is represented by a hybrid circuit $\widetilde{C}_{s_\ell+2s_{\ell+1}}$ for $\tilde{f}_{s_\ell+2s_{\ell+1}}(\alpha, \beta, \gamma, v_1, v_2, z^{[\ell]}) =$

$$\tilde{\text{equ}}(z^{[\ell]}, \alpha)\big(\tilde{\text{add}}_\ell(\alpha, \beta, \gamma)(v_1 + v_2) + \cdots + \tilde{\text{thru}}_\ell(\alpha, \beta, \gamma)\, v_1\big), \quad (20)$$

which implements (13) for inputs $v_1 = \widetilde{V}_{\ell+1}(\beta)$ and $v_2 = \widetilde{V}_{\ell+1}(\gamma)$. Here the inputs v_1, v_2 and the components of $z_\theta^{[\ell]}$ take the role of the constants $c_1, \ldots, c_m$ in the function $\tilde{f}_n$ in Figure 2. We have $s_\ell + 2s_{\ell+1}$ as n in our special summation tree protocol for the function $f_{s_\ell+2s_{\ell+1}}$ in (20). Finally, the decoder circuits $\widetilde{D}_{s_\ell+2s_{\ell+1}}$ have inputs $\alpha, \beta, \gamma, \nu, \mu, \mu'$ and implement the hybrid algebraic extension function (cf. (19)) $\widetilde{\varphi}(\alpha, \beta, \gamma, \nu, \mu, \mu') =$

$$\Big(\sum_{\theta=1}^{s_\ell+2s_{\ell+1}} (\mu = 0 \textbf{ and } \mu' = 0 \textbf{ and } \theta = \nu)\ \iota_\nu\Big) + (\nu = 0 \textbf{ and } \mu' = 0 \textbf{ and } \mu = 0)\ \widetilde{V}_{\ell+1}(\beta) + (\nu = 0 \textbf{ and } \mu' = 0 \textbf{ and } \mu = 1)\ \widetilde{V}_{\ell+1}(\gamma) + \Big(\sum_{\theta=1}^{s_\ell} (\nu = 0 \textbf{ and } \mu = 0 \textbf{ and } \theta = \mu')\ r_\theta^{[\ell]}\Big), \quad (21)$$

where ι_ν is the variable corresponding to the ν-the bits of the combined bit vector for α, β, γ. Note that μ is a single bit/variable input, which selects which of the two values $\widetilde{V}_{\ell+1}(\beta)$ or $\widetilde{V}_{\ell+1}(\gamma)$ is output by the decoder. The circuit $\widetilde{C}_{s_\ell+2s_{\ell+1}}$ for $\tilde{f}_{s_\ell+2s_{\ell+1}}$ (20) has by the polynomial-depth Assumption 2.2 for its embedded circuits for the DAG structure test functions $\tilde{\text{add}}_\ell$, etc., itself DAG structure test functions that satisfy the log-depth Assumption 2.1, and Remark 2.6 applies for the tree summation protocol. The decoder circuit $\widetilde{\varphi}$ of (21) has depth $O(\log n)$, which the verifier can evaluate for random field element inputs for the bits of α, etc., by the subprotocol (17) and a certificate for $\widetilde{V}_{\ell+1}(\vec{r}^{\,[\ell+1]})$. □

REMARK 2.8. **Space complexity for the prover.** As the sum-check protocols involved move from level ℓ to level $\ell + 1$, but the circuit evaluation from level $\ell + 1$ to level ℓ, the required circuit node values either need to be stored or re-computed when needed. This trade-off is well-studied in the field of automatic differentiation [13]. □

3. LINEAR ALGEBRA CIRCUITS

For certifying linear algebra computations, poly-log-depth uniform circuits for linear algebra replace the binary tree in Figure 2. Many classical problems, such as determinant, rank, linear system solution, etc., are in $\mathcal{NC}^2$. The determinant is for 0 or large characteristic verifiable with a prover efficient protocol of complexity $N^{\omega}(\log N)^{O(1)}$, where ω is the matrix multiplication exponent. One uses the processor-efficient circuits from [17], but with the preconditioner from [10], which uses two verifier-selected random field elements. The rank protocol uses the parallel algorithm in [23] and parallel algorithms for the characteristic polynomial [5, 7]. Possibly $O(N)$ trailing coefficients v_i are certified to be zero by verifying $0 = w = \sum_i v_i r^{i-1}$ with a verifier-selected random r of $(\log N)^{O(1)}$ bit-size. The leading Δ coefficients of the characteristic polynomial can be computed by log-squared-depth uniform circuits with $\Delta^{1/2}N^{\omega}(\log N)^{O(1)}$ work, for characteristic 0 or $\geq \Delta + 1$ [7]. For $\Delta = N$ one has the characteristic polynomial, and $\Delta \ll N$ is used for the resultant in Section 4. No processor-efficient circuit for the characteristic polynomial or prover-efficient protocol for the rank, with a fixed number of verifier-selected random elements, are known to us.

4. THE MULTIVARIATE RESULTANT

We will use the Macaulay matrices [4, 22] for our determinant circuits to compute the resultant. There exist smaller matrices which also lead to proof protocols, of possibly smaller prover complexity. Candidate matrix constructions are Dixon matrices [6, 19] and matrices for sparse systems [8, 14], to name a few of many. Here we use Canny's simple formulation as an illustration of the power of the GKR protocol and our variant.

We consider a system of k homogeneous polynomials in k variables over a field F,

$$f_i(y_1,\ldots,y_k) = \sum_{e_{i,1}+\cdots+e_{i,k}=d_i} \tilde{f}(i,e_{i,1},\ldots,e_{i,k},c_1,\ldots,c_m)\times y_1^{e_{i,1}}\cdots y_k^{e_{i,k}} \in \mathsf{F}[y_1,\ldots,y_k],\ 1\leq i\leq k; \quad (22)$$

note that we now use i for the i-th polynomial, not the input i. Let $d_{\max} = \max_{1\leq i\leq k}(d_i)$. In (22) the single function for the coefficients $\tilde{f}(\vec{X},\vec{E}_1,\ldots,\vec{E}_k,\vec{C})$ represents a $(k\log(d_{\max}))^{O(1)}$-depth algebraic extension circuit, whose inputs are the binary digits of i, the binary digits of the term exponents $e_{i,1},\ldots,e_{i,k}$, and $m = (k\log(d_{\max}))^{O(1)}$ constants c_μ from a field F (cf. Figure 1). If needed for the computation of $\tilde{f}$, a subsequence of the constants could contain the bits of the binary representation of k, d_1, …, d_k. From that subsequence of constants one can then from the input i to $\tilde{f}$ compute the binary representation of d_i by the circuit for

$$d_i = \tilde{d}(i,c_1,\ldots,c_m) = \textstyle\sum_{\kappa=1}^{k} \tilde{\mathrm{equ}}(\kappa,i)\,d_\kappa, \quad (23)$$

In (23) the bits of all κ are wired into $\tilde{d}$ from the input constants $c_1 = 0$ and $c_2 = 1$, as are the bits of d_κ. The DAG structure test functions are computable by the prover and verifier for the agreed positions of the bits of d_1, …, d_k in $c_1,\ldots,c_m$. Note that k is implicit in the node labels of the circuit $\tilde{d}$.

We assume that $\tilde{f}$ evaluates to 0 if one or more of the $e_{i,\kappa} < 0$, as we will evaluate $\tilde{f}$ with such values in (27) below. In our evaluations we will always have $d_i = e_{i,1} + \cdots + e_{i,k}$. An example for coefficients which are representable by such circuits $\tilde{f}$ was given in the abstract: $\tilde{f}(i,e_{i,1},\ldots,e_{i,k},c_1,\ldots,c_m) = (i + e_{i,1} + \cdots\ e_{i,k})!\ /\ (i!\ e_{i,1}!\cdots\ e_{i,k}!)$; see Remark 2.6 how to achieve circuit depth $(k\log(d_{\max}))^{O(1)}$.

Macaulay gives a matrix $A^{[\mathrm{num}]} \in \mathsf{F}^{D\times D}$ and a submatrix $A^{[\mathrm{den}]} \in \mathsf{F}^{D'\times D'}$ $A^{[\mathrm{num}]}$ with entries the symbolic coefficients of (22) such that the resultant of (22) is the polynomial $\det(A^{[\mathrm{num}]})/\det(A^{[\mathrm{den}]})$ (in the symbolic coefficients) evaluated at the actual coefficient values. The fundamental property is that the system (22) has a (projective) zero, which is a common root $\neq (0,\ldots,0)$, if and only if the resultant is zero. Canny handles the difficulty that $\det(A^{[\mathrm{den}]})$ may evaluate to zero at the actual coefficient values by dividing the characteristic polynomial of $A^{[\mathrm{num}]}$ by the one of $A^{[\mathrm{den}]}$, which correspond to the resultant of the polynomial system for $f_i(y_1,\ldots,y_k) - \lambda y_i^{d_i}$. We have:

$$\delta \stackrel{\mathrm{def}}{=} 1 + \textstyle\sum_{i=1}^{k}(d_i - 1),\quad D \stackrel{\mathrm{def}}{=} \binom{\delta+k-1}{k-1},$$

$$D' \stackrel{\mathrm{def}}{=} D - \Delta,\quad \Delta \stackrel{\mathrm{def}}{=} \textstyle\left(\sum_{i=1}^{k}\prod_{i'\neq i} d_{i'}\right). \quad (24)$$

Again, D are the dimensions of $A^{[\mathrm{num}]}$, D' the dimensions of $A^{[\mathrm{den}]}$ and Δ the degree of the resultant in the coefficients of (22).

Our determinant protocol can certify the resultant in verifier complexity $O((\log D)^2)$, that is, $(k\log(d_{\max}))^{O(1)}$. We first describe a circuit that computes the determinant of the larger matrix $A^{[\mathrm{num}]}$. We can wrap the circuits $\tilde{f}$ for the coefficients by circuits that with input I,J return $a^{[\mathrm{num}]}_{I,J}$, but then the verifier constructs the Macaulay matrix for each input pair (I,J). Instead, we construct a single circuit that at a certain level has D^2 nodes with the values of $a^{[\mathrm{num}]}_{I,J}$ for $\tilde{f}$ for the larger Macaulay matrix $A^{[\mathrm{num}]}$. One can consider that preparatory circuit as one decoder circuit in the sense of Figure 2.

The Macaulay construction labels the D rows of $A^{[\mathrm{num}]}$ by all terms $y_1^{\eta_1}\cdots y_k^{\eta_k}$ of total degree δ. The columns are labeled by pairs $(i, y_1^{\chi_1}\cdots y_k^{\chi_k})$ with $\chi_1 + \cdots + \chi_k = \delta - d_i$ and $\chi_{i'} \leq d_{i'} - 1$ for all $1 \leq i' \leq i-1$ (Canny calls the terms "reduced in $y_1,\ldots,y_{i-1}$"). There are exactly D such pairs, which we obtain as follows: for each term $y_1^{\eta_1}\cdots y_k^{\eta_k}$ with $\eta_1 + \cdots + \eta_k = \delta$ compute

$$i = \min\{i' \mid 1 \leq i' \leq k \text{ and } \eta_{i'} \geq d_{i'}\} \quad (25)$$

(because of the value of δ the set is non-empty) and the following corresponding term:

$$y_1^{\chi_1}\cdots y_k^{\chi_k} = y_1^{\eta_1}\cdots y_{i-1}^{\eta_{i-1}} y_i^{\eta_i - d_i} y_{i+1}^{\eta_{i+1}}\cdots y_k^{\eta_k}. \quad (26)$$

Note that the same vector $[\chi_1,\ldots,\chi_k]$ may be associated with different i's. We give an example for $k = 3$, $d_1 = 2$, $d_2 = 2$, $d_3 = 1$ with $\delta = 3$, $D = \binom{3+2}{2} = 10$ (see [3, Resultant Example]).

$\vec{\eta}$	[3,0,0]	[2,1,0]	[2,0,1]	[1,2,0]	[1,1,1]
$(i,\vec{\chi})$	(1,[1,0,0])	(1,[0,1,0])	(1,[0,0,1])	(2,[1,0,0])	(3,[1,1,0])
$\vec{\eta}$	[1,0,2]	[0,3,0]	[0,2,1]	[0,1,2]	[0,0,3]
$(i,\vec{\chi})$	(3,[1,0,1])	(2,[0,1,0])	(2,[0,0,1])	(3,[0,1,1])	(3,[0,0,2])

The entry in $A^{[\mathrm{num}]}$ for rows and columns labeled in the above manner is then $a^{[\mathrm{num}]}_{[\eta_1,\ldots,\eta_k],\,[i,\chi_1,\ldots,\chi_k]} =$

$$\tilde{f}(i,\eta_1-\chi_1,\ldots,\eta_k-\chi_k,c_1,\ldots,c_m). \quad (27)$$

Our preparatory circuit has inputs d_1, …, d_k, which are represented in binary, and the constants $c_1,\ldots,c_m$. In the hybrid version those constitute $s_{\ell_{\mathrm{inp}}} = k\lceil\log_2(d_{\max}+1)\rceil + m$ input variables. In our GKR protocol, the function $\widetilde{V}_{\ell_{\mathrm{inp}}}$ can then be evaluated by the verifier in complexity $O(k\log(d_{\max}))^{O(1)}$. We first generate on a certain level a block of D exponent vectors of all $y_1^{\eta_1}\cdots y_k^{\eta_k}$ with $\eta_1 + \cdots + \eta_k = \delta$.

1. Compute the individual exponent lists $[0,\ldots,\delta]$.
2. Combine the list to exponent vectors $[\eta_1,\ldots,\eta_{k-1}]$ with $0 \leq \eta_i \leq \delta$. There are $(\delta+1)^{k-1}$ such vectors which are produced by passthru wiring in the circuit, whose DAG structure test functions are computable by the prover and verifier.
3. Sort the vectors via a Batcher sorting network of log-squared depth, with the comparison being the circuit for comparing the total degrees $\eta_1 + \cdots + \eta_{k-1}$ in binary.
4. Truncate the sorted list at the first D nodes via wiring and DAG structure test functions.

5. Append in an additional block of binary nodes for each vector the exponent of the k-th variable $\eta_k = \delta - \eta_1 - \cdots - \eta_{k-1}$.

Note that we could have sorted the array $[\eta_1, \ldots, \eta_{k-1}]$ of node blocks with respect to a graded lexicographic order, which only would affect the sign of the computed resultant.

The column labels $(i, [\chi_1, \ldots, \chi_k])$ are computed in a second array of D node groups by filtering the first group by (25) and (26). For that, the values d_i are required as circuits with input i, which are constructed from the input values, which are made available by passthru nodes, by an expression analogous to (23). At this point one again could sort the pair of column indices lexicographically. Leaving the column labels unsorted again only affects the sign of the resultant. The entries in $A^{[\mathrm{num}]}$ are then computed in D^2 nodes on a certain level by (27) pairing each of the nodes for $[\eta_1, \ldots, \eta_k]$ with each of the nodes for $(i, [\chi_1, \ldots, \chi_k])$ by circuit wire. Note that $\tilde{f}$ receives the arguments i and $\eta_\kappa - \chi_\kappa$ from internal nodes and the c_μ from the input nodes by passthru. Those D^2 nodes constitute the input nodes to a determinant circuit of depth $O((\log D)^2)$.

The submatrix $A^{[\mathrm{den}]}$ of $A^{[\mathrm{num}]}$ is constructed by deleting rows and columns. The rows with labels $[\eta_1, \ldots, \eta_k]$ which are deleted are those which have labels that are reduced for all but one component, that is, for all those labels

$$\exists 1 \le i \le k \text{ (dep. on the label) } \forall 1 \le i' \le k, i' \ne i \colon \eta_{i'} \le d_{i'} - 1. \qquad (28)$$

By δ in (24) we then have $\eta_i \ge d_i$. It is easy to see that there are Δ in (24) many such labels. For each deleted row, a corresponding column is deleted, namely the column with the label matched to the row label as above, that is, $(i, [\chi_1, \ldots, \chi_k]) = (i, [\eta_1, \ldots, \eta_{i-1}, \eta_i - d_i, \eta_{i+1}, \ldots, \eta_k])$. In $A^{[\mathrm{num}]}$, we have placed the coefficient of $y_i^{d_i}$ of f_i, namely, $\tilde{f}(i, 0, \ldots, 0, d_i, 0, \ldots, 0, c_1, \ldots, c_m)$ in the entry in the row and column thus identified. In the example above, there are $D' = 2 = 10 - (2 \cdot 2 + 2 \cdot 1 + 2 \cdot 1)$ remaining rows labeled [2,0,1] and [0,2,1] and two remaining columns labeled by the corresponding pairs (1,[0,0,1]) and (2,[0,0,1]) that constitute $A^{[\mathrm{den}]}$. For $k = 2$ we have $\delta = d_1 + d_2 - 1$, $D = \binom{\delta+1}{1} = d_1 + d_2$, and $\Delta = d_1 + d_2$, which means that all rows and columns are removed and there is no $A^{[\mathrm{den}]}$ matrix. The matrix $A^{[\mathrm{num}]}$ then is the classical matrix associated with the Sylvester resultant. Our protocol is still remarkable in that it certifies the Sylvester resultant, that is relative primeness of two univariate polynomials, in $(\log(d_1 + d_2))^{O(1)}$ verifier complexity.

We can compute an array of row labels of the submatrix $A^{[\mathrm{den}]}$ by filtering the list of row labels of $A^{[\mathrm{num}]}$ by condition (28), placing a high degree label where the condition fails, and then sorting the list. The column labels for $A^{[\mathrm{den}]}$ are computed exactly as for $A^{[\mathrm{num}]}$ and the entries $a_{I,J}^{[\mathrm{den}]}$ the same way as well.

Following Canny [4], the resultant is found as the constant coefficient of the polynomial $h(\lambda) = \det(A^{[\mathrm{num}]} - \lambda I)/\det(A^{[\mathrm{den}]} - \lambda I)$ of degree Δ. We compute h using the $\mathcal{NC}$ circuits of Section 3 for the characteristic polynomials, then perform a division [24]. Canny [4] suggests to compute h from the $\Delta + 1$ higher degree coefficients of above characteristic polynomials. As seen in previous section this can be achieved with circuits of size $O(\Delta^{\frac{1}{2}} D^\omega)$ if F has characteristic 0 or $\ge \Delta + 1$. For the field $\mathsf{F} = \mathbb{Q}$ and entries of bit length $D^{O(1)}$, as is the case for our example in the abstract, the verifier first selects a random prime p of length $(\log D)^{O(1)}$ and executes the interactive protocol in $\mathsf{F} = \mathbb{Z}_p$. The initial constants c_μ of $\tilde{f}$ in that situation must be computable modulo p in $(\log D)^{O(1)}$ bit complexity.

Acknowledgement: Supported in part by OpenDreamKit (EU) Grant 676541 (Dumas), NSF (USA) Grant CCF-1421128 (Kaltofen) and NNSF (China) Grant 11571350 (Zhi).

Note added on 6/23/17: 'bit length $D^{O(1)}$'↔'bit length $(\log D)^{O(1)}$' 5 lines from end.

REFERENCES

[1] P. M. Beame, S. A. Cook, and H. J. Hoover. 1986. Log depth circuits for division and related problems. *SIAM J. Comput.* 15 (1986), 994–1003. https://doi.org/10.1137/0215070

[2] R. P. Brent and H. T. Kung. 1982. A regular layout for parallel adders. *IEEE Trans. Computers* C-31, 3 (1982), 260–264. https://doi.org/10.1109/TC.1982.1675982

[3] J. Canny, E. Kaltofen, and Yagati Lakshman. 1989. Solving systems of non-linear polynomial equations faster. In *Proc. 1989 ISSAC (ISSAC'89)*. ACM Press, New York, N. Y., 121–128. https://doi.org/10.1145/74540.74556 URL: http://www.math.ncsu.edu/~kaltofen/bibliography/89/CKL89.pdf.

[4] John F. Canny. 1990. Generalised Characteristic Polynomials. *J. Symb. Comput.* 9, 3 (1990), 241–250. https://doi.org/10.1016/S0747-7171(08)80012-0

[5] A. L. Chistov. 1985. Fast parallel calculation of the rank of matrices over a field of arbitrary characteristic. In *Proc. FCT '85 (LNCS)*, Vol. 199. Springer, Germany, 63–69. https://doi.org/10.1007/BFb0028792

[6] Arthur D. Chtcherba and Deepak Kapur. 2006. Conditions for Determinantal Formula for Resultant of a Polynomial System. In *Proc. 2006 ISSAC*. ACM, New York, NY, USA, 55–62. https://doi.org/10.1145/1145768.1145784

[7] L. Csanky. 1976. Fast parallel matrix inversion algorithms. *SIAM J. Comput.* 5, 4 (1976), 618–623. https://doi.org/10.1137/0205040

[8] Carlos D'Andrea. 2002. Macaulay style formulas for sparse resultants. *Trans. Amer. Math. Soc.* 354, 7 (2002), 2595–2629. https://doi.org/10.1090/S0002-9947-02-02910-0

[9] Jean-Guillaume Dumas and Erich L. Kaltofen. 2014. Essentially Optimal Interactive Certificates In Linear Algebra. In *Proc. 39th ISSAC 2014*, Katsusuke Nabeshima (Ed.). ACM, New York, N. Y., 146–153. https://doi.org/10.1145/2608628.2608644 URL: http://www.math.ncsu.edu/~kaltofen/bibliography/14/DuKa14.pdf.

[10] Jean-Guillaume Dumas, Erich Kaltofen, Emmanuel Thomé, and Gilles Villard. 2016. Linear Time Interactive Certificates for the Minimal Polynomial and the Determinant of a Sparse Matrix. In *Proc 2016 ISSAC*, Markus Rosenkranz (Ed.). ACM, New York, N. Y., 199–206. https://doi.org/10.1145/2930889.2930908 URL: http://www.math.ncsu.edu/~kaltofen/bibliography/16/DKTV16.pdf.

[11] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. 2008. Delegating computation: interactive proofs for muggles. In *STOC 2008*, Cynthia Dwork (Ed.). ACM Press, 113–122. https://doi.org/10.1145/1374376.1374396

[12] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. 2015. Delegating Computation: Interactive Proofs for Muggles. *J. ACM* 62, 4 (2015), 27:1–27:64. https://doi.org/10.1145/2699436

[13] A. Griewank. 1992. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods & Software* 1 (1992), 35–54. https://doi.org/doi:10.1080/10556789208805505

[14] J.P Jouanolou. 1991. Le formalisme du résultant. *Advances in Mathematics* 90, 2 (1991), 117 – 263. https://doi.org/10.1016/0001-8708(91)90031-2

[15] Erich Kaltofen and Pascal Koiran. 2005. On the complexity of factoring bivariate supersparse (lacunary) polynomials. In *Proc. 2005 ISSAC*, Manuel Kauers (Ed.). ACM Press, New York, N. Y., 208–215. https://doi.org/10.1145/1073884.1073914 ACM SIGSAM's ISSAC 2005 Distinguished Paper Award. URL: http://www.math.ncsu.edu/~kaltofen/bibliography/05/KaKoi05.pdf.

[16] Erich L. Kaltofen, Michael Nehring, and B. David Saunders. 2011. Quadratic-Time Certificates in Linear Algebra. In *Proc. 2011 ISSAC*, Anton Leykin (Ed.). ACM, New York, N. Y., 171–176. https://doi.org/10.1145/1993886.1993915 URL: http://www.math.ncsu.edu/~kaltofen/bibliography/11/KNS11.pdf.

[17] E. Kaltofen and V. Pan. 1991. Processor efficient parallel solution of linear systems over an abstract field. In *Proc. SPAA '91 3rd Ann. ACM Symp. Parallel Algor. Architecture*. ACM Press, New York, N.Y., 180–191. https://doi.org/10.1145/113379.113396 URL: http://www.math.ncsu.edu/~kaltofen/bibliography/91/KaPa91.pdf.

[18] E. Kaltofen and B. Trager. 1990. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *JSC* 9, 3 (1990), 301–320. https://doi.org/10.1016/S0747-7171(08)80015-6 URL: http://www.math.ncsu.edu/~kaltofen/bibliography/90/KaTr90.pdf.

[19] Deepak Kapur, Tushar Saxena, and Lu Yang. 1994. Algebraic and Geometric Reasoning Using Dixon Resultants. In *Proc. ISSAC '94*. ACM, New York, NY, USA, 99–107. https://doi.org/10.1145/190347.190372

[20] Christoph Koutschan and Thotsaporn Thanatipanonda. 2013. Advanced Computer Algebra for Determinants. *Annals of Combinatorics* 17, 3 (2013), 509–523. http://www.risc.jku.at/people/ckoutsch/det/

[21] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. 1992. Algebraic Methods for Interactive Proof Systems. *J. ACM* 39, 4 (Oct. 1992), 859–868. https://doi.org/10.1145/146585.146605

[22] F. S. Macaulay. 1916. Algebraic theory of modular systems. Cambridge Tracts 19, Cambridge. (1916).

[23] Ketan Mulmuley. 1987. A fast parallel algorithm to compute the rank of a matrix over an arbitrary field. *Combinatorica* 7, 1 (1987), 101–104. https://doi.org/10.1007/BF02579205

[24] M. Sieveking. 1972. An algorithm for division of power series. *Computing* 10 (1972), 153–156.

[25] Justin Thaler. 2013. Time-Optimal Interactive Proofs for Circuit Evaluation. In *CRYPTO '13 (LNCS)*, Ran Canetti and Juan A. Garay (Eds.), Vol. 8043. Springer, 71–89. https://doi.org/10.1007/978-3-642-40084-1_5