CHICAGO JOURNAL OF THEORETICAL COMPUTER SCIENCE 2019, Article 02, pages 1–19
http://cjtcs.cs.uchicago.edu/

Non-commutative computations: lower bounds and polynomial identity testing*

Guillaume Lagarde   Guillaume Malod   Sylvain Perifel

Received January 16, 2017; Revised August 31, 2019; Published September 10, 2019

Abstract: In the setting of non-commutative arithmetic computations, we define a class of circuits that generalize algebraic branching programs (ABP). This model is called *unambiguous* because it captures the polynomials in which all monomials are computed in a similar way (that is, all the *parse trees* are isomorphic).

We show that unambiguous circuits of polynomial size can compute polynomials that require ABPs of exponential size, and that they are incomparable with skew circuits.

Generalizing a result of Nisan [23] on ABPs, we provide an exact characterization of the complexity of any polynomial in our model, and use it to prove exponential lower bounds for explicit polynomials such as the determinant.

Finally, we give a white-box deterministic polynomial-time algorithm for polynomial identity testing (PIT) on unambiguous circuits over $\mathbb{R}$ and $\mathbb{C}$.

1 Introduction

Arithmetic circuits as a model for complexity-theoretic questions has enjoyed an increase of interest over the last ten years. This is due in particular to a general strategy, called *geometric complexity theory*, to tackle the main open question of complexity, P versus NP, via algebraic means (see the survey [7] or the website [16]). One of its intermediate goals is to prove that computing the permanent cannot be efficiently reduced to computing the determinant. This question can naturally be seen as an analogue

*This work was supported by ANR CompA (ANR-13-BS02-0001-01)

Key words and phrases: arithmetic complexity, lower bounds, non-commutative computations, polynomial identity testing

 DOI: 10.4086/cjtcs.2019.002

of P versus NP when using arithmetic circuits as model of computation, as introduced by Valiant in founding articles [29, 30]. Interest in arithmetic circuits was also sparked by a string of applications of a measure based on *partial derivatives* (see the surveys [28, 9]). Recent generalizations of this technique, coupled with strong parallelization results for arithmetic circuits, have brought us closer to showing that the permanent cannot be written as a small determinant (see the survey [26]).

One of the earlier articles using such a notion of partial derivatives is Nisan [23], which studies computations in the non-commutative ring $\mathbb{F}\langle X\rangle$: variables do not commute so that xy and yx are distinct monomials. Studying non-commutative computations is an important endeavour, as they arise naturally (for instance when computing over matrices), but also because they can have applications for *commutative* computations (see [10, 5], in particular the use of non-commutative determinants to approximate the commutative permanent). Moreover, non-commutativity is one kind of restriction that can be imposed on general arithmetic computations. Others, such as multilinearity, have yielded stronger lower bounds than in the general case (see for example [24], again using partial derivatives). Nisan [23] again provides an early example, proving exponential lower bounds for non-commutative arithmetic formulas and more generally for non-commutative algebraic branching programs in 1991. However this did not lead to superpolynomial lower bounds for general non-commutative *circuits*. Very little progress was made for a long time, and there is still no known lower bound for general non-commutative arithmetic circuits that is stronger than those that we already have for general *commutative* arithmetic circuits. Recently, Hrubeš, Wigderson, and Yehudayoff [17] suggested a new line of attack on the general arithmetic circuit lower bound question, linking it to the classical *Sum-of-squares* problem. Finally, Nisan's result was extended in [21] to a more powerful model, so-called *skew circuits*, arithmetic circuits where every multiplication involves at most one argument which is not a variable or a constant. There, non-commutative skew circuits were shown to be exponentially more powerful than non-commutative branching programs, but exponentially less powerful than general non-commutative circuits.

Here, we again extend Nisan's result but in a different direction. Given a (non-commutative) circuit, we can look at the set of monomials it produces (before any grouping/cancellations). If we pretend that the computation is also non-associative, the monomial comes with parentheses to indicate the "way" in which it was computed. The pattern of parentheses for a given monomial (the structure of the monomial in a sense) can also be seen as a tree. We will focus on circuits where this structure or tree is the same for all the monomials computed by the circuit, and we will call these circuits *unambiguous*. If one computes an algebraic branching program as a circuit, then monomials are all obtained by successive multiplication on the right, and they all have the same structure. Our model is thus more general than the one considered by Nisan. Perhaps the most striking aspect of Nisan's paper, more than its elegance, is that it goes much farther than the usual results in complexity, which try to get, *for a specific polynomial*, either a lower bound or an upper bound, *with big O notation*, hopefully *asymptotically* matching. In contrast, Nisan gives an *exact* expression for the complexity of *any* polynomial. More precisely, the minimal size of a branching program computing a polynomial f is expressed via the ranks of a family of matrices defined by f, for all branching programs in a certain "canonical" form. We prove a generalization of his theorem, characterizing the minimal size of a "canonical" unambiguous circuit computing any polynomial f, also in terms of ranks of matrices. This exact characterization also yields exponential lower bounds, making unambiguous circuits another "strongest" model of non-commutative computation for which we have superpolynomial lower bounds (it is incomparable with the models of [21], see Section 5).

Finally we consider the problem of *Polynomial Identity Testing* (PIT) for our model. In a general setting, PIT asks whether a given circuit computes the zero polynomial. The Schwartz-Zippel Lemma [11, 31, 27] yields a simple and efficient randomized algorithm: evaluate the circuit at a random point and answer "non-zero" iff the result was non-zero. Finding a deterministic algorithm ("derandomizing PIT") would imply circuit lower bounds [18], making the search for such an algorithm an important open problem.

In the non-commutative setting there is also a polynomial-time randomized algorithm [6] (for *polynomial-degree* circuits only). But here derandomization has some significant results. A first efficient white-box[1] deterministic algorithm for non-commutative ABPs was given by Raz & Shpilka [25]. Here we use ideas from a simpler construction given by Arvind et al. [1, 2] to get a polynomial-time deterministic PIT algorithm for unambiguous circuits over $\mathbb{R}$ or $\mathbb{C}$.

Since the publication of this article there have been several works extending its results. Lagarde et al. [20] extended the lower bound to circuits with an exponential number of different associative structures in its polynomials and the PIT result to a sum of a constant number of unambiguous circuits, with possibly different associative structures. A connection with language theory, automata and formal series was then noticed in [13]. Nisan's result is a simple consequence of the characterization of recognizable formal series on words by Fliess [15] (see also [8]). As explained in [14], an extension of [15] to formal series on trees yields a Nisan-like result for nonassociative noncommutative circuits. Based on this, [14] recovers and extends previous lower bounds in the associative setting, in particular for noncommutative circuits with slightly less than all possible parse trees. In both characterization results, the minimal size is given by the rank of the so-called Hankel matrix. The Hankel matrix and its links with automata and ABPs were also used in [19] to prove learnability results for arithmetic circuits.

2 Non-commutative computations, parse trees and unambiguous circuits

We consider *non-commutative* computations (over a field $\mathbb{F}$ and a set X of variables): the variables do not commute (that is we do not have $xy = yx$). Nevertheless addition is still commutative and the rules for the constants do not change, according to the underlying field $\mathbb{F}$. We can therefore think of monomials as strings over the alphabet X. The ring of non-commutative polynomials over a field $\mathbb{F}$ and a set X of variables is denoted by $\mathbb{F}\langle X\rangle$. We will use the following convenient notation for polynomials of $\mathbb{F}\langle x_1,\dots,x_n\rangle$: $P(x_1,\dots,x_n) = \sum_{\bar{x}} a_{\bar{x}}\bar{x}$, where $\bar{x}$ denotes a monomial and $a_{\bar{x}} \in \mathbb{F}$ the corresponding coefficient in P.

As a basic model of computation, we use arithmetic circuits (see the survey [28]): that is, directed acyclic graphs in which vertices of indegree zero are called input gates; all the other vertices are labeled with $+$ or $\times$; and the unique vertex of outdegree zero is called the output gate. We add the following important points:

- input gates are only labeled with variables $x \in X$, not constants;
- multiplication gates have fan-in two, their inputs are ordered and the multiplication is interpreted according to this order (the left child is multiplied before the right child);

[1] A PIT algorithm is *white-box* if it can use the structure of the computation model; it is *black-box* if it only requires an evaluation oracle.

- addition gates have unbounded fan-in and perform a linear combination of their inputs, with the associated coefficients $\alpha_i \in \mathbb{F}$ being given on the edges.

Let us emphasize that an addition gate can possibly have only one input, thus performing a scalar multiplication. The polynomial computed by each gate is defined inductively in a natural way.

Nisan [23] studied non-commutative computations, mainly concentrating on *algebraic branching programs* (ABP). An ABP is a directed acyclic graph A with two distinguished vertices s (source) and t (target), such that every arc is labeled with a constant $\alpha \in \mathbb{F}$ or a variable $x \in X$. The weight of a path in A is the monomial equal to the product of the labels of the arcs in the path. The polynomial computed by A is then the sum of the weights of all paths from s to t in A. This computation model is at least as powerful as formulas (and indeed strictly stronger in the multilinear commutative setting, see [12]), and at most as powerful as general circuits.

Actually, simulating an ABP by an arithmetic circuit yields a *skew* circuit, i.e., a circuit where, for every multiplication gate, at least one of its arguments is an input gate. Indeed, if we build the circuit inductively, to obtain a gate computing the same polynomial as a vertex in the ABP we just need to multiply previously obtained polynomials *on the right* and by variables or constants. So the resulting circuit is not only skew, but every "right" argument of a multiplication gate is an input gate: let us call such a circuit *right-skew*. We will now describe more precisely the way monomials are obtained using the notion of parse trees from [22].

Definition 2.1. The set of parse trees of a circuit $\mathcal{C}$ is defined by induction on its size:

- if $\mathcal{C}$ is of size 1 it has only one parse tree: itself;
- if the output gate of $\mathcal{C}$ is a $+$-gate whose arguments are the gates α and β, the parse trees of $\mathcal{C}$ are obtained by taking either a parse tree of the subcircuit rooted at α and the arc from α to the output or a parse tree of the subcircuit rooted at β and the arc from β to the output;
- if the output gate of $\mathcal{C}$ is a $\times$-gate whose arguments are the gates α and β, the parse trees of $\mathcal{C}$ are obtained by taking a parse tree of the subcircuit rooted at α, a parse tree of a disjoint copy of the subcircuit rooted at β, and the arcs from α and β to the output.

A parse tree $\mathcal{T}$ computes a polynomial $\mathrm{val}(\mathcal{T})$ in a natural way: this is the monomial equal to the product of the variables labeling the leaves of $\mathcal{T}$ (from left to right), multiplied by the coefficient equal to the product of the constants labeling the edges pointing to a $+$-gate. So parse trees are in one-to-one correspondence with the monomials computed by the circuit (before regrouping), and summing the values of the parse trees thus yields the computed polynomial.

It is easy to see that the parse trees of a right-skew circuit are all in the shape of a comb. In other words, any monomial, say $x_{i_1} \cdots x_{i_d}$, is computed in the following way: $((((x_{i_1}x_{i_2})x_{i_3}) \cdots x_{i_d})$. This "comb" shape is exactly like the paths of an ABP, and the two models are basically identical.

These ideas were used in [21] to obtain lower bounds for skew circuits, not just right-skew (or left-skew by symmetry). Part of the intuition explaining the weakness of such circuits is that, although parse trees do not have the shape of a comb any longer, they are still like paths: they are trees but at each branching one of the branches stops immediately. Although one cannot use the same ideas as in

Nisan's case directly, this "path" structure of parse trees means that the degree of the monomial is built up incrementally, so that we can pinpoint exactly the gate where a specific degree is reached in all parse trees.

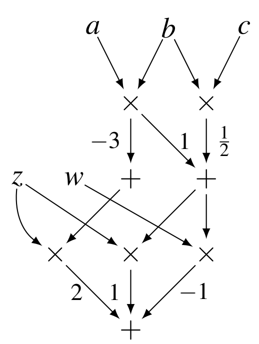
Instead of focusing on this incremental nature of the parse trees of right-skew circuits, we can also note that they all have the same "shape". This is not true of general skew circuits, for instance if we have just a sum of a right-skew circuit and a left-skew circuit. The circuits we will be interested in are those where all parse trees have the same shape, not necessarily a comb or a path as in the case of skew circuits, but a general tree (see Figures 1 and 2).

Definition 2.2 (Unambiguous circuits). Two parse trees $\mathcal{T}$ and $\mathcal{T}'$ are *isomorphic* if there is a bijection f from the vertices of $\mathcal{T}$ to the vertices of $\mathcal{T}'$ such that:

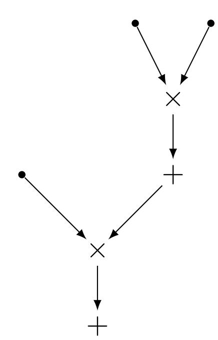
1. leaves are sent to leaves, $+$-gates to $+$-gates, $\times$-gates to $\times$-gates;
2. there is an arc from u to v iff there is one from $f(u)$ to $f(v)$;
3. the order of arguments for $\times$-gates is preserved: if u is the left argument and v the right argument of a $\times$-gate w, then $f(u)$ is the left argument and $f(v)$ the right argument of $f(w)$.

A circuit is called *unambiguous* if all its parse trees are isomorphic. The isomorphism class is called the *shape* of the circuit.

Note that because there are no constants on the leaves, an unambiguous circuit computes a homogeneous polynomial at each gate. In particular, the output is a homogeneous polynomial.



(a) An unambiguous circuit C.



(b) The shape of the circuit C.

Figure 1: An unambiguous circuit and its shape. Note: the output gate is drawn at the bottom.

Let us emphasize that the class of polynomials computable by unambiguous circuits of polynomial size is quite large and natural: it contains all the ABPs as already explained (cf. Figure 2), as well as for instance the palindrome polynomial (cf. Section 5) used in [23, 21]. It is rich enough to contain, for all k, the polynomial f_k (defined in [21]) which requires exponential-size circuits of skew-depth k, thus creating a hierarchy of increasing power inside general non-commutative circuits. A final example: the $\Theta(n2^n)$ computation of the permanent, tersely explained by Nisan in [23], is also unambiguous and is asymptotically as fast as Ryser's formula (but has the advantage of being monotone and non-commutative).

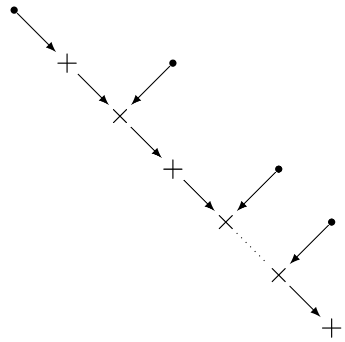


Figure 2: Shape of an ABP turned into an unambiguous circuit.

Let us now focus on our first goal: generalizing Nisan's result and giving a characterization of the size of unambiguous circuits necessary to compute a given polynomial. To get his results, Nisan focused on ABPs with a specific structure (homogeneous, layered, with linear forms on the edges).

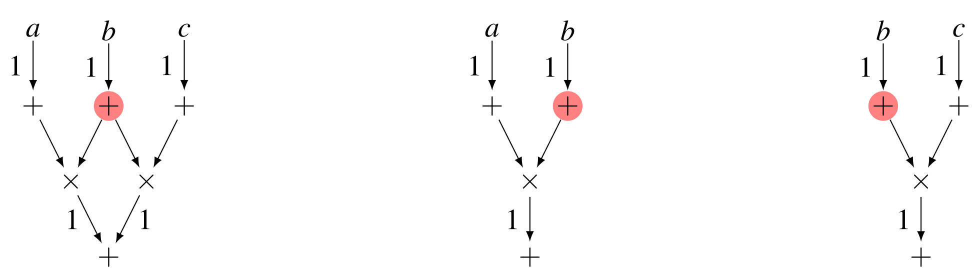
In our case also, we will need such a canonical form for unambiguous circuits.

Definition 2.3 (Canonical form for unambiguous circuits)**.** A circuit $\mathcal{C}$ is canonical if:

1. $\mathcal{C}$ is unambiguous.

2. $\mathcal{C}$ is layered, starting with input gates, then $+$-gates, then $\times$-gates, alternating until a final $+$-gate. $+$-gates at a given layer can only use $\times$-gates from the previous layer as arguments, while $\times$-gates at a given layer must use $+$-gates as arguments, at least one of which is from the previous layer.

3. Each $+$-gate has a unique position in the shape. That is, each $+$-gate appears at most once in any parse tree and, for any two parse trees $\mathcal{T}$ and $\mathcal{T}'$ containing an addition gate u, the isomorphism from $\mathcal{T}$ to $\mathcal{T}'$ maps u to u (see Figure 3).



(a) An unambiguous circuit C not in canonical form.

(b) A parse tree of C with the red gate on the right.

(c) Another parse tree of C with the red gate on the left.

Figure 3: An unambiguous circuit violating Condition 3 of the definition of canonical form.

Any unambiguous circuit can be rendered canonical at a small cost, as shown in the lemma below.

Lemma 2.4. *Given an unambiguous circuit $\mathcal{C}$ of degree d and size s, it is possible to construct in polynomial time a canonical unambiguous circuit $\mathcal{C}'$ of size at most $2ds$ computing the same polynomial.*

The proof of Lemma 2.4 relies on a careful inspection of the proof of [22, Lemma 2]. A circuit is called *multiplicatively disjoint* if each $\times$-gate has disjoint subcircuits as inputs. The result [22, Lemma 2] states that every circuit C of degree d can be turned efficiently into an equivalent multiplicatively disjoint circuit of size $(|C|+d)^{O(1)}$.

The *formal degree* of a gate is the degree of the polynomial computed by this gate if no cancellation would occur, that is:

- the formal degree of an input gate (labeled with a variable) is 1;
- the formal degree of a $+$-gate $g = \sum \alpha_i g_i$ is the maximum of the formal degrees of the gates g_i;
- the formal degree of a $\times$-gate $g = g_1 g_2$ is the sum of the formal degrees of the gates g_1 and g_2.

Proof of Lemma 2.4. Condition (2) is easy to obtain. If the output of a $\times$-gate is the input of another $\times$-gate, just add a useless $+$-gate between the two. If the output of an addition gate g_1 is the input of some $+$-gates $g_2, \dots, g_k$, delete g_1 and add to the inputs of $g_2, \dots, g_k$ the previous inputs of g_1 with the associated linear coefficients. Thus, condition (2) is obtained at the cost of a blow-up of factor at most 2.

Condition (3) is obtained by applying the algorithm to transform a general circuit into a multiplicatively disjoint circuit from [22, Lemma 2]. The resulting circuit has size $\leq 2ds$. For the sake of completeness, we recall the construction here (modified a little bit for the needs of non-commutativity).

For each gate $\alpha \in \mathcal{C}$ of formal degree e, the new circuit $\mathcal{C}'$ contains distinct gates $\alpha_1, \alpha_2, \dots, \alpha_{d+1-e}$. α_k is called a clone of index k of α. In $\mathcal{C}$, if α is a $\times$-gate of formal degree e with left input β of formal degree e_1 and right input γ of formal degree e_2, then in $\mathcal{C}'$, α_k has left input β_k and right input γ_{k+e_1}. In $\mathcal{C}$, if α is a $+$-gate of formal degree e with inputs $\beta^1, \beta^2, \dots, \beta^j$ with coefficients $c_1, c_2, \dots, c_j$, then, in $\mathcal{C}'$, α_k has inputs $\beta_k^1, \beta_k^2, \dots, \beta_k^j$ with coefficients $c_1, \dots, c_j$.

The fact that $\mathcal{C}'$ is multiplicatively disjoint is immediate with the following "index property": in $\mathcal{C}'$, the gates in the subcircuit rooted at α_k of formal degree e are clones whose indices lie between k and $k+e-1$. We can easily prove this property by induction. Suppose α_k is a multiplication gate (the other case being simpler) of degree e with left input β_k of degree e_1 and right input γ_{k+e_1} of degree $e-e_1$. Then, by induction hypothesis, the gates in the subcircuit rooted at β_k are clones whose indices lie between k and $k+e_1-1$ and gates in the subcircuit rooted at γ_{k+e_1} are gates whose indices lie between $k+e_1$ and $k+e_1+(e-e_1)-1 = k+e-1$. This concludes this case and the proof.

We prove by contradiction that $\mathcal{C}'$ respects condition (3). Let α_j be an addition gate in $\mathcal{C}'$ and $\mathcal{T}$ and $\mathcal{T}'$ two parse trees which contain α_j but at two different positions. Let $l_1, l_2, \dots, l_a$ (resp. $g_1, g_2, \dots, g_b$) be the unique path in $\mathcal{T}$ (resp. $\mathcal{T}'$) from the output gate to α_k (thus $l_a = g_b = \alpha_k$). Because α_k does not share the same position in the two parse trees, it means that there is a minimal c such that l_c and g_c are $+$-gates with different positions. It means that l_{c-1} and g_{c-1} are two $\times$-gates (because the circuit is constituted of alternating layers) and that l_c and g_c are inputs of l_{c-1} and g_{c-1}, one as left input, one as right input (let us say in that order). As the circuit is unambiguous, l_c and g_c must be of same degree e. g_{c-1} and l_{c-1} are clones of same index because the path from the output gate to these gates are identical. Let us say they are of index k. Thus l_c is a clone of index k and g_c is a clone of index $k+e$ (because of the construction and the fact that one is a left input, the other a right input of the multiplication gate). Thanks to the index property, this means that the subcircuits defined by l_c and g_c are clones whose index lies between k and $k+e-1$ for l_c and between $k+e$ and $k+2e-1$ for g_c. These two sub-circuits are thus disjoint, but this in contradiction with the fact that α_j belongs to both of them. □

3 Decomposition lemma for canonical unambiguous circuits

Nisan observed that if P has an ABP of small size, then, for all i, P can be decomposed as a small sum of polynomials of the form $g \cdot h$ where g and h are homogeneous polynomials of respective degrees i and $(d-i)$.

This is a common step in lower bound proofs: writing any computation in the model under consideration as a small sum of "building blocks" for which some complexity measure is very low. Here we extend Nisan's decomposition to canonical unambiguous circuits.

Because the position of each $+$-gate in the shape is unique, we can associate to each $+$-gate α a unique *type* $(i,p) \in \mathbb{N}^2$ which encodes the position of the addition gate in the shape. For that we need to define the *degree* of a gate γ in a shape: this is merely the number of leaves in the subtree rooted at γ (thus, in any parse tree, a gate which corresponds to γ in the shape computes, if it does not vanish, a monomial of this precise degree).

Definition 3.1 (Type of a gate)**.** Let α be an addition gate: it corresponds to an addition gate γ in the shape. Let i be the degree of γ. If L is the unique path (in the shape) from γ to the output gate, we denote by $\beta_1,\dots,\beta_k$ the gates (in the shape) which appear as left input of a $\times$-gate of L. Let p be the sum of the degrees of the β_i. Then, the *type* of α is (i,p).

Intuitively, i is the degree computed by α and p is the degree of the monomials which are concatenated on the left in computations involving α (see Figure 5). In order to state our decomposition result we need a definition from [21] which we restate here.

Definition 3.2 (j-products, see Figure 4)**.** Given homogeneous polynomials $g,h \in \mathbb{F}\langle X\rangle$ of degrees d_g and d_h respectively and an integer $j \in [0,d_h]$, we define the *j-product of g and h* — denoted $g \times_j h$ — as follows:

- When g and h are monomials, then we can factor h uniquely as a product of two monomials h_1h_2 such that $\deg(h_1) = j$ and $\deg(h_2) = d_h - j$. In this case, we define $g \times_j h$ to be $h_1 \cdot g \cdot h_2$.

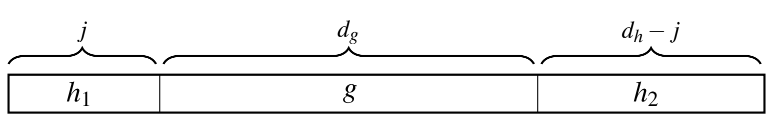
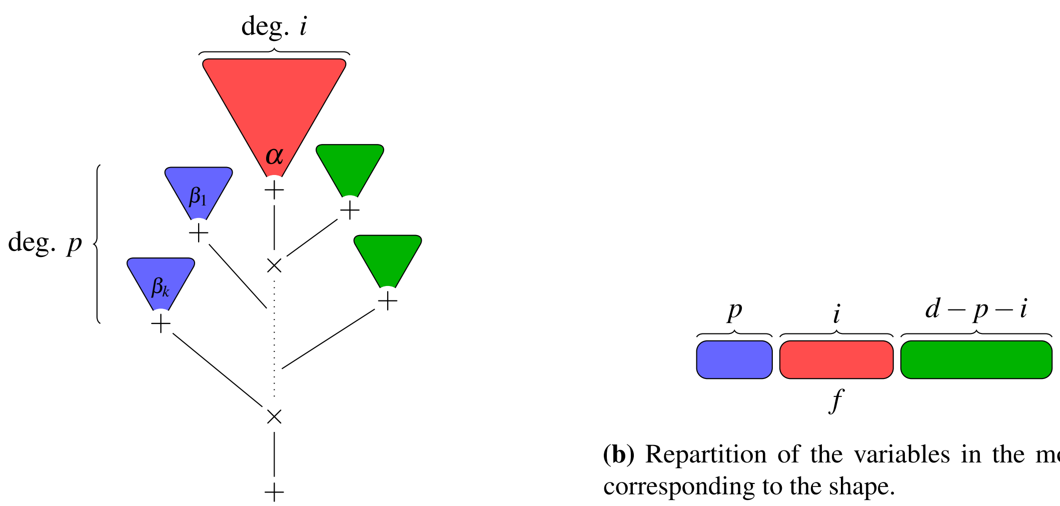


Figure 4: j-product of two monomials g and h.

- The map is extended bilinearly to general homogeneous polynomials g,h. Formally, let g,h be general homogeneous polynomials, where $g = \sum_\ell g_\ell$, $h = \sum_i h_i$ and g_ℓ, h_i are monomials of g,h respectively. For $j \in [0,d_h]$, each h_i can be factored uniquely into h_i^1, h_i^2 such that $\deg(h_i^1) = j$ and $\deg(h_i^2) = d_h - j$. And $g \times_j h$ is defined to be $\sum_i \sum_\ell h_i^1 g_\ell h_i^2$.

Proposition 3.3 (Decomposition for canonical unambiguous circuits)**.** *If a polynomial P of degree d is computed by a canonical unambiguous circuit and if (i, p) is an existing type*[2] *of addition gate, then P can be written as $P = \sum_{j=1}^{k_{i,p}} f_j \times_p h_j$, where:*

1. *$k_{i,p}$ is the number of addition gates of type (i, p) and $f_1, \ldots, f_{k_{i,p}}$ are the polynomials computed by these gates;*
2. *$\forall j, \deg(f_j) = i$ and $\deg(h_j) = d - i$.*



(a) A shape and a gate α of type (i, p).

(b) Repartition of the variables in the monomial corresponding to the shape.

Figure 5: Type of a gate in a shape.

Proof. (See Figure 5 for an illustration.) Let $\mathcal{C}$ be a canonical unambiguous circuit computing the polynomial P. We have: $P = \sum_{\mathcal{T} \in S} \mathrm{val}(\mathcal{T})$, where S is the set of all parse trees of $\mathcal{C}$. Let $\alpha_1, \ldots, \alpha_{k_{i,p}}$ be the gates of type (i, p) in $\mathcal{C}$, computing respectively the polynomials $f_1, f_2, \ldots, f_{k_{i,p}}$. By definition of the type, the f_j are of degree i. For $1 \le j \le k_{i,p}$, let S_j be the set of parse trees containing the gate α_j. Because a parse tree contains at most one addition gate of a given type, and because the type of a $+$-gate is unique, we have $S_j \cap S_k = \emptyset$ for $j \neq k$. Moreover, if a given type exists, every parse tree contains an addition gate of this type because the considered circuit is unambiguous. Thus the S_j are a partition of S: $S = S_1 \sqcup S_2 \sqcup \cdots \sqcup S_{k_{i,p}}$, where $\sqcup$ denotes the disjoint union. We can then rewrite the previous equality as $P = \sum_{\mathcal{T} \in S} \mathrm{val}(\mathcal{T}) = \sum_{j=1}^{k_{i,p}} \sum_{\mathcal{T} \in S_j} \mathrm{val}(\mathcal{T})$.

Fix $j \in [1, k_{i,p}]$. Consider the circuit $C_j(y)$ obtained by changing α_j into an input gate labeled with a new variable y and deleting unused gates. Note that $C_j(f_j) = C$ (abusing notations and using the name of the circuit for the computed polynomial). Let T_j be the set of parse trees of C_j containing the input gate α_j. The value of any parse tree $\mathcal{T} \in T_j$ is of the form $y \times_p h_{\mathcal{T}}$ where $h_{\mathcal{T}}$ is a monomial of degree

[2]That is, at least one addition gate is of this type.

$(d-i)$. Then, by bilinearity of the j-product, $V_j(y) := \sum_{\mathcal{T}\in T_j} \mathrm{val}(\mathcal{T}) = y \times_p h_j$, where h_j is a polynomial of degree $(d-i)$. Note that $\sum_{\mathcal{T}\in S_j} \mathrm{val}(\mathcal{T}) = V_j(f_j)$ and therefore $\sum_{\mathcal{T}\in S_j} \mathrm{val}(\mathcal{T}) = f_j \times_p h_j$. □

4 Exact complexity for canonical unambiguous circuits

We will use the number of $+$-gates of a canonical unambiguous circuit as an estimate of its size. The following lemma shows that this is a good measure of overall size.

Lemma 4.1. *Let $\mathcal{C}$ be a canonical unambiguous circuit with s $+$-gates. Then we can transform $\mathcal{C}$ into a new canonical unambiguous circuit, without changing the shape, with s $+$-gates and at most s^2 $\times$-gates.*

Proof. Denote by s_i the number of $+$-gates on the i-th layer of $\mathcal{C}$. If $\mathcal{C}$ has strictly more than s^2 $\times$-gates, then one layer i contains strictly more than s_i^2 $\times$-gates. It means that two different $\times$-gates on the same layer perform the same computation; therefore one of them can be deleted and its output replaced by the output of the other one. □

We will use this notion of size to get an exact expression of the complexity of computing a given polynomial with a canonical unambiguous circuit. To do this, we create a complexity measure which is an extension for canonical unambiguous circuits of the one given by Nisan [23] for algebraic branching programs. For a given homogeneous polynomial P of degree d and each integer $i \le d$, Nisan defined the *partial derivative matrix* $M^{(i)}(P)$, which is a $n^{d-i} \times n^i$ matrix whose rows are indexed by monomials on X of degree $(d-i)$ and columns by monomials of degree i. The entry (m_1, m_2) of the matrix is defined to be the coefficient of the monomial $m_1 m_2$ in P. Intuitively speaking, the rank of the matrix $M^{(i)}(P)$ is a measure of how "correlated" the prefix of length i of a monomial appearing in P is to the rest of the monomial. Small ABPs have "information bottlenecks" at each degree i, and hence the amount of correlation in the computed polynomial must be low. In our case the correlation will be between the prefix of degree p and the suffix of degree $(d-p-i)$ on the one hand, and the middle part of degree i on the other hand.

Definition 4.2. Let P be a polynomial of degree d on n variables $(x_1, x_2, \ldots, x_n)$. For $(i,p) \in [0,d] \times [0,d]$ with $i + p \le d$, we define $M^{(i,p)}(P)$ to be a matrix of size $n^{d-i} \times n^i$. Rows are indexed by all pairs $(\bar{x}, \bar{z}) \in \{x_1, \ldots, x_n\}^p \times \{x_1, \ldots, x_n\}^{d-p-i}$. Columns are indexed by words $\bar{y} \in \{x_1, \ldots, x_n\}^i$. Finally, $M^{(i,p)}(P)_{(\bar{x},\bar{z}),\bar{y}}$ is the coefficient of the monomial $\bar{x} \cdot \bar{y} \cdot \bar{z}$ in P.

We can now express exactly the number of additions needed to compute a given polynomial by a canonical unambiguous circuit.

Theorem 4.3. *Let P be a homogeneous polynomial of degree d and $\mathcal{T}$ a shape with d leaves. Then the minimal number of addition gates needed to compute P by a canonical unambiguous circuit with shape $\mathcal{T}$ is exactly equal to $\sum_{(i,p)\in S} \mathrm{rank}\left(M^{(i,p)}(P)\right)$, where S is the set of all existing types of $+$-gates in the shape $\mathcal{T}$.*

Proof. Fix a canonical unambiguous circuit $\mathcal{C}$ with shape $\mathcal{T}$ which computes P. Fix also (i,p) — an existing type of addition gate — and let $\alpha_1, \ldots, \alpha_{k_{i,p}}$ be all the (i,p)-addition gates in $\mathcal{C}$. Let $P = \sum_{j=1}^{k_{i,p}} f_j \times_p h_j$ be the decomposition given by Proposition 3.3. To simplify notations, set also $k = k_{i,p}$.

Decomposition of the matrix $M^{(i,p)}$ as $L^{(i,p)}R^{(i,p)}$. We show that $M^{(i,p)}$ is the product of two "small" matrices $L^{(i,p)}$ and $R^{(i,p)}$:

- $R^{(i,p)}$ is a matrix of size $k \times n^i$. Rows are indexed by all gates $\alpha_1, \ldots, \alpha_k$. Columns are indexed by monomials $\bar{y} \in \{x_1, \ldots, x_n\}^i$. $R^{(i,p)}_{t,\bar{y}}$ is the coefficient of the monomial $\bar{y}$ in the polynomial f_t computed by the gate α_t.
- $L^{(i,p)}$ is a matrix of size $n^{d-i} \times k$. Rows are indexed by all pairs $(\bar{x}, \bar{z}) \in \{x_1, \ldots, x_n\}^p \times \{x_1, \ldots, x_n\}^{d-p-i}$. Columns are indexed by all gates $\alpha_1, \ldots, \alpha_k$. $L^{(i,p)}_{(\bar{x},\bar{z}),t}$ is the coefficient of the monomial $\bar{x}\bar{z}$ in the polynomial computed by the circuit where α_t is replaced by an input gate with value 1. That is: $L^{(i,p)}_{(\bar{x},\bar{z}),t}$ is the coefficient of the monomial $\bar{x}\bar{z}$ in the polynomial h_t.

One can easily verify that $M^{(i,p)} = L^{(i,p)}R^{(i,p)}$.

Lower bound. Since $\operatorname{rank}\left(M^{(i,p)}\right) \leq \operatorname{rank}\left(L^{(i,p)}\right) \leq k$, the number k of addition gates of type (i,p) must be at least $\operatorname{rank}(M^{(i,p)})$. Therefore, considering all existing types, we have just proved that the number of addition gates is at least $\sum_{(i,p)\in S} \operatorname{rank}\left(M^{(i,p)}(P)\right)$.

Upper bound. We prove that if $\operatorname{rank}\left(M^{(i,p)}\right) < k$, we can delete one (i,p)-addition gate in the circuit. We will possibly be increasing at the same time the number of $\times$-gates but, thanks to Lemma 4.1, this is innocuous. If $\operatorname{rank}\left(L^{(i,p)}\right) = \operatorname{rank}\left(R^{(i,p)}\right) = k$, then, since $L^{(i,p)}$ and $R^{(i,p)}$ are $n^{d-i} \times k$ and $k \times n^i$ matrices, respectively, $\operatorname{rank}\left(M^{(i,p)}\right)$ should also be k. Thus, either $L^{(i,p)}$ or $R^{(i,p)}$ is of rank strictly less than k.

If $\operatorname{rank}\left(R^{(i,p)}\right) < k$, then one row (let us say, w.l.o.g., the first row) of $R^{(i,p)}$ is a linear combination of the other rows. Going back to the meaning of the matrix, it means that the polynomial f_1 computed by the gate α_1 is a linear combination of the polynomials $f_2, \ldots, f_k$ computed by the gates $\alpha_2, \ldots, \alpha_k$. Let us say $f_1 = \sum_{i=2}^k c_i f_i$ for $c_i \in \mathbb{F}$. We construct a new circuit where α_1 is deleted. We denote by $\beta_1, \ldots, \beta_m$ the $\times$-gates which receive as input α_1. In the new circuit, we create $(k-1)$ copies of $\beta_1, \ldots, \beta_m$ — namely $\beta_1^2, \ldots, \beta_m^2, \beta_1^3, \ldots \beta_m^3, \ldots, \beta_1^k, \ldots, \beta_m^k$. β_j^i does exactly the same computation as β_j, but instead of taking α_1 as input, it takes α_i. Finally, an addition gate in the old circuit which took as input a β_j now takes $\sum_{i=2}^k c_i \beta_j^i$ as input.

If $\operatorname{rank}\left(L^{(i,p)}\right) < k$, then one column (let us say, w.l.o.g., the first column) of $L^{(i,p)}$ is a linear combination of the other columns. This means that there are constants $c_2, \ldots, c_k$ such that $h_1 = \sum_{j=2}^k c_j h_j$. Let $\gamma_1, \ldots, \gamma_m$ be all the coefficients on the input edges of α_1 coming respectively from multiplication gates $\beta_1, \ldots, \beta_m$. In the new circuit, we delete α_1 and we add for all $1 \leq l \leq m, 2 \leq j \leq k$ an edge between β_l and α_j with the coefficient $c_j \gamma_l$. The new circuit computes the polynomial $\sum_{j=2}^k (f_j + c_j f_1) \times_p h_j$. By bilinearity of the j-product, this is equal to

$$\begin{aligned}
&\sum_{j=2}^k (f_j \times_p h_j + (c_j f_1) \times_p h_j) &&= \sum_{j=2}^k f_j \times_p h_j \quad + \quad \sum_{j=2}^k (c_j f_1) \times_p h_j \\
=\; &\sum_{j=2}^k f_j \times_p h_j \quad + \quad \sum_{j=2}^k f_1 \times_p (c_j h_j) &&= \sum_{j=2}^k f_j \times_p h_j \quad + \quad f_1 \times_p \left(\sum_{j=2}^k (c_j h_j)\right) \\
=\; &\sum_{j=2}^k f_j \times_p h_j \quad + \quad f_1 \times_p h_1 &&= P. \qquad \square
\end{aligned}$$

Remark 4.4. When the shape is right-skew (thus corresponding to an ABP), then $p=0$ in the proof above, and $M^{(i,p)}$ is the usual matrix $M^{(i)}$ of Nisan [23]. Since the number of additions gates in the shape corresponds exactly to the number of vertices in an ABP in canonical form, our result is a direct extension of Nisan's.

5 Comparison with skew circuits.

In this section we show that the classes of polynomials computed by polynomial-size unambiguous circuits on the one hand, and by polynomial-size skew circuits on the other hand, are incomparable. Define the palindrome of degree d over n variables as:

$$Pal^d(x_1,\dots,x_n) := \sum_{\bar{z}\in\{x_1,..,x_n\}^{d/2}} \bar{z}.\bar{z}_m,$$

where $\bar{z}_m$ is the mirror of $\bar{z}$ (e.g $\bar{z}_m = x_3x_2x_1$ if $\bar{z} = x_1x_2x_3$). It is easy to construct a small unambiguous and skew circuit for $Pal^d(x_1,\dots,x_n)$ by using the following inductive formula:

$$Pal^d(x_1,\dots,x_n) = \sum_i^n x_i Pal^{d-2}(x_1,\dots,x_n)x_i.$$

We can then use the construction for $Pal^d(x_1,\dots,x_n)$ to compute the square of the palindrome with a small unambiguous circuit. Note that [21] shows that the square of the palindrome polynomial needs exponential-size skew circuits: therefore, unambiguous is not included in skew (when considering polynomial-size circuits).

In the remainder of this section we construct a polynomial computable by a skew circuit of polynomial size but not by unambiguous circuits of polynomial size. The idea is the following: given a canonical unambiguous circuit of degree d (without any condition on its shape), there is always an addition gate of type (i,p) where $i\in[\frac{d}{3},\frac{2d}{3}], p\in[0,d-i]$ (Lemma 5.1, proof given in the appendix). We then consider a polynomial such that the associated matrices $M^{(i,p)}$ have an exponential rank for all $i\in[\frac{d}{3},\frac{2d}{3}], p\in[0,d-i]$. According to the previous section, this means that computing the polynomial by unambiguous circuits requires at least an exponential number of gates.

Lemma 5.1. *Given a canonical unambiguous circuit computing a polynomial of degree d, there is always an existing type (i,p) where $i\in[\frac{d}{3},\frac{2d}{3}], p\in[0,d-i]$.*

Proof. It is sufficient to prove that there is a $+$-gate of degree $i\in[\frac{d}{3},\frac{2d}{3}]$: the condition on p follows immediately from the definition of the type. Let α be a $\times$-gate of degree $>\frac{2}{3}d$ as close as possible to the leaves. Let β,γ be the inputs of α and i,j their respective degree. We have $i+j>\frac{2d}{3}, 1\le i\le\frac{2d}{3}, 1\le j\le\frac{2d}{3}$. These conditions force i or j to be in $[\frac{d}{3},\frac{2d}{3}]$. ☐

The moving palindrome of degree n on $(n+1)$ variables is:

$$Pal^n_{mov}(x_1,\dots,x_n,w) := \sum_{l\in[0,\frac{2n}{3}]} w^l Pal^{\frac{n}{3}}(x_1,\dots,x_n)w^{\frac{2n}{3}-l},$$

where w is a fresh variable (distinct from the x_i). The first proposition below is easy to prove and follows from the construction for $Pal^d(x_1,\dots,x_n)$, the second is an application of our size characterization for canonical unambiguous circuits.

Proposition 5.2. *$Pal^n_{mov}(x_1,\dots,x_n,w)$ is computable by a skew circuit of size polynomial in n.*

Proposition 5.3. *Computing $Pal^n_{mov}(x_1,\dots,x_n,w)$ with a canonical unambiguous circuit requires at least $n^{n/6}$ gates.*

Proof. Consider a canonical unambiguous circuit C computing Pal^n_{mov}. Thanks to Lemma 5.1, we know that there is always an existing type (i,p) where $i \in [\frac{n}{3},\frac{2n}{3}], p \in [0,n-i]$. To apply Theorem 4.3, it is enough to show that for all such (i,p), $\mathrm{rank}(M^{(i,p)}(Pal^n_{mov})) \geq n^{n/6}$. This will possible because for each such type, there is a polynomial in the sum defining Pal^n_{mov} which has a large rank and the other polynomials will not interfere.

Let us fix a particular $(i,p), i \in [\frac{n}{3},\frac{2n}{3}], p \in [0,n-i]$. Because $i \leq \frac{2n}{3}$ we have $p+(n-p-i) \geq \frac{n}{3}$. Then one of the two following cases occurs.
Case $p \geq \frac{n}{6}$. In this case we show first that

$$\mathrm{rank}(M^{(i,p)}(Pal^n_{mov})) \geq \mathrm{rank}(M^{(i,p)}(w^{p-\frac{n}{6}}Pal^{\frac{n}{3}}(x_1,\dots,x_n)w^{n-p-\frac{n}{6}})).$$

Indeed,

$$M^{(i,p)}(Pal^n_{mov}) = \sum_{l\in[0,\frac{2n}{3}]} M^{(i,p)}(w^l Pal^{\frac{n}{3}}(x_1,\dots,x_n)w^{\frac{2n}{3}-l});$$

And note then that, if (a,b) is a coordinate of a non-zero coefficient of

$$M^{(i,p)}(w^{p-\frac{n}{6}}Pal^{\frac{n}{3}}(x_1,\dots,x_n)w^{n-p-\frac{n}{6}})$$

and (a',b') is a coordinate of a non-zero coefficient of

$$M^{(i,p)}(w^l Pal^{\frac{n}{3}}(x_1,\dots,x_n)w^{\frac{2n}{3}-l}),$$

with $l \neq p-\frac{n}{6}$, then $a \neq a'$ and $b \neq b'$. Finally, observe that in this case, every row and column of $M^{(i,p)}(w^{p-\frac{n}{6}}Pal^{\frac{n}{3}}(x_1,\dots,x_n)w^{n-p-\frac{n}{6}})$ contains at most one non-zero coefficient and there are exactly $n^{n/6}$ non-zero coefficients. Thus:

$$\mathrm{rank}\left(M^{(i,p)}(Pal^n_{mov})\right) \geq \mathrm{rank}\left(M^{(i,p)}(w^{p-\frac{n}{6}}Pal^{\frac{n}{3}}(x_1,\dots,x_n)w^{n-p-\frac{n}{6}})\right) \geq n^{n/6}.$$

Case $n-p-i \geq \frac{n}{6}$. With similar arguments, we have this time

$$\mathrm{rank}\left(M^{(i,p)}(Pal^n_{mov})\right) \geq \mathrm{rank}\left(M^{(i,p)}(w^{p+i-\frac{n}{6}}Pal^{\frac{n}{3}}(x_1,\dots,x_n)w^{\frac{5n}{6}-p-i})\right) \geq n^{n/6}. \quad \square$$

6 Lower bounds for permanent and determinant

In the non-commutative setting we need to define an order on the variables of each monomial of the permanent or the determinant. We will start by considering the so-called Cayley permanent and determinant:

$$\mathrm{per}_n^C = \sum_{s \in S_n} \prod_{i=1}^{n} x_{1,s(1)} \cdots x_{n,s(n)} \text{ and } \det_n^C = \sum_{s \in S_n} \mathrm{sgn}(s) \prod_{i=1}^{n} x_{1,s(1)} \cdots x_{n,s(n)}.$$

To get lower bounds we need to estimate the ranks of certain matrices $M^{(i,p)}$. The following lemma is proved exactly in the same way as Lemma 2 in [23].

Lemma 6.1. *For all $i \leq n$, $p \leq n-i$,* $\mathrm{rank}\left(M^{(i,p)}(\mathrm{per}_n^C)\right) = \mathrm{rank}\left(M^{(i,p)}(\det_n^C)\right) = \binom{n}{i}$.

We can now obtain the following lower bounds, thanks to Lemma 5.1.

Theorem 6.2. *Computing* per_n^C *or* $\det_n^C$ *with an unambiguous circuit requires at least* $\binom{n}{n/3}$ *gates.*

For other orders on the monomials we once again follow Nisan.

Definition 6.3. Two polynomials P and Q are called weakly equivalent if for each monomial of P with non-zero coefficient there exists a monomial of Q with the same variables (but perhaps in a different order) with non-zero coefficient, and vice-versa.

Lemma 6.4. *For all $i \leq n$, $p \leq n-i$,* $\mathrm{rank}\left(M^{(i,p)}\right)$ *for any polynomial weakly equivalent to the permanent or the determinant is at least* $\binom{n}{i}$.

Theorem 6.5. *Computing a polynomial weakly equivalent to* per_n^C *or* $\det_n^C$ *with an unambiguous circuit requires at least* $\binom{n}{n/3}$ *gates.*

7 Polynomial Identity Testing via Hadamard product

Here, we give a deterministic polynomial-time algorithm for *PIT* for the polynomials computed by unambiguous non-commutative circuits.[3] We will use the following binary operation over polynomials from [1].

Definition 7.1. Given two polynomials $P = \sum_{\bar{x}} a_{\bar{x}} \bar{x}$ and $Q = \sum_{\bar{x}} b_{\bar{x}} \bar{x}$, the *Hadamard product* of P and Q, written $P \odot Q$, equals $\sum_{\bar{x}} a_{\bar{x}} b_{\bar{x}} \bar{x}$.

In [1], a logspace algorithm is given which, on input two ABPs A and B, outputs a new ABP C computing the Hadamard product of the polynomials computed by A and B. Consequently, they observed that this result gives the following derandomization for PIT.

[3] A deterministic polynomial-time algorithm for PIT on non-commutative skew circuits is claimed in [4], but in fact it only works for circuits that are *both* skew and unambiguous (private communication with the authors). Actually, this algorithm seems to be removed from the conference version [3].

Theorem 7.2 ([1])**.** *The problem of polynomial identity testing for non-commutative algebraic branching programs over $\mathbb{R}$ is in* P.

Here, we extend this result: we give a construction to perform the Hadamard product of two unambiguous circuits with the same shape. In other words, we prove that the class of unambiguous circuits of a given shape is *stable under Hadamard product*. As in the case of ABPs, it will provide a deterministic polynomial-time algorithm for PIT over unambiguous circuits.

W.l.o.g. we work with homogeneous polynomials. Indeed, if P and Q are decomposed into homogeneous components $P = \sum_{k=1}^{n} P_i$ and $Q = \sum_{k=1}^{n} Q_i$, then $P \odot Q = \sum_{k=1}^{n} P_i \odot Q_i$. Circuits will be assumed canonical, since Lemma 2.4 gives an explicit algorithm working in polynomial time to transform an unambiguous circuit into its canonical form. The idea is to create a circuit computing iteratively the Hadamard product of all pairs of addition gates of same type. The regularity of the parse tree will allow us to spread the Hadamard product layer by layer.

Lemma 7.3. *Let $d, d' \in \mathbb{N}$ and let $(P_i)_{1\leq i\leq n}$ and $(Q_i)_{1\leq i\leq m}$ be families of polynomials with $\deg(P_i) = d$ and $\deg(Q_i) = d'$. Set also $(\alpha_{i,j})_{1\leq i\leq n, 1\leq j\leq m} \in \mathbb{R}^{nm}$ and $(\beta_{i,j})_{1\leq i\leq n, 1\leq j\leq m} \in \mathbb{R}^{nm}$. Then:*

$$\left(\sum_{(i,j)} \alpha_{i,j} P_i Q_j\right) \odot \left(\sum_{(i,j)} \beta_{i,j} P_i Q_j\right) = \sum_{(i,j),(k,l)} \alpha_{i,j}\beta_{k,l}(P_i \odot P_k)(Q_j \odot Q_l).$$

Theorem 7.4 (Hadamard product of two unambiguous circuits)**.** *Let $\mathcal{C}$ and $\mathcal{D}$ be two unambiguous circuits in canonical form, of the same shape, and of size s and s', that compute two polynomials P and Q. Then $P \odot Q$ is computed by an unambiguous circuit of size at most ss'; moreover, this circuit can be constructed in polynomial time.*

Proof. The new circuit computes the Hadamard product of all pairs $(\alpha_1, \alpha_2) \in \mathcal{C} \times \mathcal{D}$ of addition gates of the same type. As the output gate in $\mathcal{C}$ and in $\mathcal{D}$ are of the same type[4], the new circuit will in particular compute the Hadamard product of P and Q. If the degree of α_1 and α_2 is 1, then the Hadamard product is trivial since the gates compute variables.

Assume we have constructed the circuit until layer i (that is, for each gate of degree less than or equal to i). We now show how to construct the layer $(i+1)$. Let $\alpha_1 \in \mathcal{C}$ and $\alpha_2 \in \mathcal{D}$ be two addition gates of degree $(i+1)$ and of same type. Because the circuits are unambiguous, α_1 (resp. α_2) computes a polynomial of the form $R_1 = (\sum_{(i,j)} \alpha_{i,j} P_i Q_j)$ (resp. $R_2 = (\sum_{(i,j)} \beta_{i,j} P_i Q_j)$), where the P_i are all of identical types, and where the Q_j are also all of identical types. Lemma 7.3 then shows how to compute $R_1 \odot R_2$ from the previously computed $P_i \odot P_j$ and $Q_i \odot Q_j$.

By induction, we thus construct the desired circuit layer by layer. Given a type, if there were i (resp. j) addition gates of this type in $\mathcal{C}$ (resp. in $\mathcal{D}$), we have created exactly ij gates in the new circuit. Therefore, the total number of gates in the new circuit is no more than ss'. □

Corollary 7.5. *There is a deterministic polynomial-time algorithm for PIT for polynomials computed by non-commutative unambiguous circuits over $\mathbb{R}$.*

[4]because $\mathcal{C}$ and $\mathcal{D}$ have the same shape

Proof. Given $P(x_1,\dots,x_n)$ computed by a unambiguous circuit, construct the circuit which computes $(P\odot P)(x_1,\dots,x_n)$ and evaluate it on $(1,1,\dots,1)$. The output is the sum of the squares of the coefficients of P, therefore it is equal to 0 if and only if P is equal to the zero polynomial. □

Remark 7.6. From a circuit computing a polynomial $P=\sum_{\bar{x}} a_{\bar{x}}\bar{x}$ over $\mathbb{C}$, it is not hard to deduce a circuit for the conjugate $\bar{P}=\sum_{\bar{x}} \bar{a}_{\bar{x}}\bar{x}$. Therefore, a similar algorithm works over $\mathbb{C}$, since $(P\odot\bar{P})=\sum_{\bar{x}}|a_{\bar{x}}|^2\bar{x}$.

We also obtain another corollary that is to be compared with the results of Section 6.

Corollary 7.7. *Over $\mathbb{R}$, in the non-commutative setting, computing the determinant with an unambiguous circuit is as hard as computing the permanent.*

Proof. Observe that $\det\odot\det=\mathrm{per}$. Therefore, by Theorem 7.4, from a circuit computing the determinant, we can build in polynomial time a circuit computing the permanent. □

References

[1] VIKRAMAN ARVIND, PUSHKAR S. JOGLEKAR, AND SRIKANTH SRINIVASAN: Arithmetic circuits and the Hadamard product of polynomials. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, pp. 25–36, 2009. [doi:10.4230/LIPIcs.FSTTCS.2009.2304] 3, 14, 15

[2] VIKRAMAN ARVIND, PARTHA MUKHOPADHYAY, AND SRIKANTH SRINIVASAN: New results on noncommutative and commutative polynomial identity testing. *Computational Complexity*, 19(4):521–558, 2010. [doi:10.1007/s00037-010-0299-8] 3

[3] VIKRAMAN ARVIND AND S. RAJA: The complexity of bounded register and skew arithmetic computation. In ZHIPENG CAI, ALEX ZELIKOVSKY, AND ANU G. BOURGEOIS, editors, *Computing and Combinatorics - 20th International Conference, COCOON 2014, Atlanta, GA, USA, August 4-6, 2014. Proceedings*, volume 8591 of *Lecture Notes in Computer Science*, pp. 572–583. Springer, 2014. 14

[4] VIKRAMAN ARVIND AND S. RAJA: The complexity of two register and skew arithmetic computation. *Electronic Colloquium on Computational Complexity (ECCC)*, 21:28, 2014. 14

[5] A. BARVINOK: New Permanent Estimators via Non-Commutative Determinants. *ArXiv Mathematics e-prints*, July 2000. [arXiv:math/0007153] 2

[6] ANDREJ BOGDANOV AND HOETECK WEE: More on noncommutative polynomial identity testing. In *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, CCC '05, pp. 92–99, Washington, DC, USA, 2005. IEEE Computer Society. [doi:10.1109/CCC.2005.13] 3

[7] PETER BÜRGISSER, J. M. LANDSBERG, LAURENT MANIVEL, AND JERZY WEYMAN: An overview of mathematical issues arising in the geometric complexity theory approach to $\mathsf{VP}\neq\mathsf{VNP}$. *SIAM J. Comput.*, 40(4):1179–1209, August 2011. [doi:10.1137/090765328] 1

[8] J.W. CARLYLE AND A. PAZ: Realizations by stochastic finite automata. *Journal of Computer and System Sciences*, 5(1):26 – 40, 1971. [doi:https://doi.org/10.1016/S0022-0000(71)80005-3] 3

[9] XI CHEN, NEERAJ KAYAL, AND AVI WIGDERSON: Partial derivatives in arithmetic complexity and beyond. *Foundations and Trends in Theoretical Computer Science*, 6(1-2):1–138, 2011. 2

[10] STEVE CHIEN, LARS EILSTRUP RASMUSSEN, AND ALISTAIR SINCLAIR: Clifford algebras and approximating the permanent. *J. Comput. Syst. Sci.*, 67(2):263–290, 2003. 2

[11] RICHARD A. DEMILLO AND RICHARD J. LIPTON: A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978. [doi:10.1016/0020-0190(78)90067-4] 3

[12] ZEEV DVIR, GUILLAUME MALOD, SYLVAIN PERIFEL, AND AMIR YEHUDAYOFF: Separating multilinear branching programs and formulas. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pp. 615–624, 2012. [doi:10.1145/2213977.2214034] 4

[13] NATHANAËL FIJALKOW, GUILLAUME LAGARDE, AND PIERRE OHLMANN: Tight bounds using hankel matrix for arithmetic circuits with unique parse trees. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:38, 2018. 3

[14] NATHANAËL FIJALKOW, GUILLAUME LAGARDE, PIERRE OHLMANN, AND OLIVIER SERRE: Lower bounds for arithmetic circuits via the hankel matrix. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:180, 2018. 3

[15] MICHEL FLIESS: Matrices de Hankel. *J. Math. Pures Appl. (9)*, 53:197–222, 1974. 3

[16] GCT publications. `http://gct.cs.uchicago.edu/`. 1

[17] PAVEL HRUBES, AVI WIGDERSON, AND AMIR YEHUDAYOFF: Non-commutative circuits and the sum-of-squares problem. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:21, 2010. 2

[18] VALENTINE KABANETS AND RUSSELL IMPAGLIAZZO: Derandomizing polynomial identity tests means proving circuit lower bounds. *Comput. Complex.*, 13(1/2):1–46, December 2004. [doi:10.1007/s00037-004-0182-6] 3

[19] ADAM KLIVANS AND AMIR SHPILKA: Learning restricted models of arithmetic circuits. *Theory of Computing*, 2(10):185–206, 2006. [doi:10.4086/toc.2006.v002a010] 3

[20] GUILLAUME LAGARDE, NUTAN LIMAYE, AND SRIKANTH SRINIVASAN: Lower bounds and pit for non-commutative arithmetic circuits with restricted parse trees. *computational complexity*, 09 2018. [doi:10.1007/s00037-018-0171-9] 3

[21] NUTAN LIMAYE, GUILLAUME MALOD, AND SRIKANTH SRINIVASAN: Lower bounds for non-commutative skew circuits. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:22, 2015. 2, 4, 5, 8, 12

[22] GUILLAUME MALOD AND NATACHA PORTIER: Characterizing Valiant's algebraic complexity classes. *J. Complexity*, 24(1):16–38, 2008. [doi:10.1016/j.jco.2006.09.006] 4, 7

[23] NOAM NISAN: Lower bounds for non-commutative computation (extended abstract). In *Proceedings of the 23rd ACM Symposium on Theory of Computing, ACM Press*, pp. 410–418, 1991. 1, 2, 4, 5, 10, 12, 14

[24] RAN RAZ: Multi-linear formulas for permanent and determinant are of super-polynomial size. *J. ACM*, 56(2), 2009. [doi:10.1145/1502793.1502797] 2

[25] RAN RAZ AND AMIR SHPILKA: Deterministic polynomial identity testing in non-commutative models. *Comput. Complex.*, 14(1):1–19, April 2005. [doi:10.1007/s00037-005-0188-8] 3

[26] RAMPRASAD SAPTHARISHI: Recent progress on arithmetic circuit lower bounds. *Bulletin of the EATCS*, 114, 2014. 2

[27] J. T. SCHWARTZ: Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, October 1980. [doi:10.1145/322217.322225] 3

[28] AMIR SHPILKA AND AMIR YEHUDAYOFF: Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010. 2, 3

[29] L. G. VALIANT: Completeness Classes in Algebra. In *STOC '79: Proceedings of the eleventh annual ACM symposium on Theory of computing*, pp. 249–261, New York, NY, USA, 1979. ACM Press. 2

[30] L. G. VALIANT: Reducibility by Algebraic Projections. In *Logic and Algorithmic: an International Symposium held in honor of Ernst Specker*, volume 30 of *Monographies de l'Enseignement Mathémathique*, pp. 365–380, 1982. 2

[31] RICHARD ZIPPEL: Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposiumon on Symbolic and Algebraic Computation*, EUROSAM '79, pp. 216–226, London, UK, UK, 1979. Springer-Verlag. 3

AUTHORS

Guillaume Lagarde
KTH Royal Institute of Technology,
School of Electrical Enginering and Computer Science.
glagarde@kth.se
https://guillaume-lagarde.github.io/

Guillaume Malod
Université de Paris,
IMJ-PRG, CNRS,
F-75013 Paris, France.
malod@math.univ-paris-diderot.fr

Sylvain Perifel
Université de Paris,
IRIF, CNRS,
F-75013 Paris, France. sylvain.perifel@irif.fr
https://www.irif.fr/~sperifel/