Electronic Colloquium on Computational Complexity, Report No. 115 (2026)

A Lower Bound for Read-Once Parity Branching Programs

Ben Lee Volk*

Abstract

We prove an $\tilde{\Omega}(n^2)$ lower bound for read-once parity branching programs computing an explicit boolean function on n variables. The previous best lower bound was $\tilde{\Omega}(n^{1.5})$. Our lower bound is proved by reducing the problem to a lower bound in algebraic circuit complexity.

1 Introduction

Algebraic complexity is a beautiful and mathematically rich area that studies the complexity of symbolic computation of polynomials. Virtually all the known algorithms for algebraic problems (such as computing the determinant or permanent, multiplying matrices, or computing the discrete Fourier transform) are naturally modeled using algebraic models. One of its raisons d'être, however, is also the hope that lower bounds in the algebraic model will inspire lower bounds in the arguably more natural, and definitely more common, *boolean* models of computation. A long line of work on lower bounds for algebraic models has had numerous successes, such as, to give a non-exhaustive list, super-polynomial lower bounds for monotone circuits [JS82], non-commutative formulas [Nis91], multilinear formulas [Raz09, RY09], and bounded-depth circuits [LST25, For24]; and super-linear lower bounds for circuits [Str73, BS83], algebraic branching programs and formulas [CKSV22, Kal85]. More comprehensive surveys of lower bounds in algebraic complexity are [Sap15, SY10].

These lower bounds use the *syntactic* nature of the computation. For some, it is not clear what the analogous boolean model is, and for some the corresponding lower bounds for boolean models have been in fact known even earlier.

Motivated by considerations from proof complexity, there has been some work on *functional* lower bounds for algebraic circuits [GR00, FSTW21, FKS16, HLT24]. These are lower bounds for algebraic models that do not apply only to a single polynomial, but rather to a set of polynomials all computing the same function over some limited domain.

In this paper, we give an instance in which one can prove a lower bound on a bona fide boolean model of computation by reducing to a lower bound on an algebraic model of computation. One of the main obstacles to obtaining lower bounds on boolean circuits using lower bounds on algebraic circuits is that boolean circuits can exploit boolean identities that do not hold in the algebraic setting. One can trivially convert a boolean circuit C computing a function $g : \{0,1\}^n \to \{0,1\}$ to an algebraic circuit over $\mathbb{F}_2$ gate-by-gate (say by replacing AND gates with multiplication gates and NOT gates with gates that add, modulo 2, the boolean value '1'), and the resulting algebraic circuit computes a polynomial that agrees with g on the boolean cube. But its specific form depends on the circuit C: as a trivial example, the boolean function $g(x) = x$ is functionally identical to the function $g(x) = x \wedge x$, but the straightforward way alluded to above for converting a boolean circuit computing $x \wedge x$ to a polynomial would result

*Efi Arazi School of Computer Science, Reichman University, Israel. Email: benleevolk@gmail.com. The research leading to these results has received funding from the Israel Science Foundation (grant number 843/23).

ISSN 1433-8092

in the polynomial x^2, which is distinct from the polynomial x. Therefore, a lower bound on algebraic circuits computing a specific polynomial doesn't rule out the possibility that there's a different efficient way to compute the same function over the boolean domain.

The driving force behind our method is that some boolean models of computation yield *multilinear* polynomials when one applies the natural transformation that "algebrizes" them. Since two multilinear polynomials that agree on $\mathbb{F}_2^n$ are identical, we can deduce exactly which polynomial is obtained after this transformation, and prove (syntactic) lower bounds for this polynomial. The easy proofs for these observations appear in Section 2.

1.1 Read-Once Parity Branching Programs

The model we consider is the following:

Definition 1.1. *A read-once parity branching program ($\oplus$-BP) is a directed, acyclic multigraph with a source node s and a target node t. Every edge in the graph is labeled either by a constant in $\{0,1\}$, a variable x_i or a negated variable $\neg x_i$. On every $s \to t$ path, every variable appears at most once. The program accepts an input x if the number of $s \to t$ paths consistent with x is odd. The* size *of the program is the number of edges.*

Parity branching programs have been considered as a natural extension of deterministic branching programs, and a natural variant of non-deterministic branching programs (see Part V of the book [Juk12] for a thorough survey on this area). In the deterministic and non-deterministic models, exponential lower bounds for read-once branching programs have been known for decades [Zák84, BHST87, Weg88] and there are even lower bounds for branching programs that are allowed to read every input at most k times along every path for any constant k [Oko91, BRS93, Tha98]. Note that in the definition above the restriction is syntactic: we require that on every path, every variable appears at most once. The "semantic" model that only imposes this condition on paths consistent with some input has also been considered in the branching program literature (in [Juk12] it is called *weakly* read-once, and it is shown to be exponentially more powerful than the syntactic model).

However, the lower bounds for the deterministic and non-deterministic models do not apply in the parity branching program model. Jukna [Juk12, Research Problem 16.14] explicitly poses the question of proving exponential lower bounds for read-once $\oplus$-BPs. Jukna proves an exponential lower bound when the branching program is *oblivious*. An oblivious read-once $\oplus$-BP is a read-once branching program which is also layered, and in every layer all edges are labeled using the same variable. Prior to this work, the best lower bound for read-once $\oplus$-BPs was $\Omega(n^{3/2}/\log n)$, for the element distinctness function, which follows by adapting Nečiporuk's [Nec66] method to this model (the argument appears in [KW93], where it is attributed to Pudlák). This lower bound in fact holds for general parity branching programs, even without the read-once restriction. Cheraghchi, Hirahara, Myrisiotis and Yoshida proved a similar lower bound for the meta-complexity problem MKTP [CHMY24], and this model was also studied by Homeister [SS05, Hom06, BHW03], who proved lower bounds in some restricted settings.

Our main result is an almost quadratic lower bound for read-once parity branching programs.

Theorem 1.2. *There exists an explicit family of functions $\{f_n : \{0,1\}^n \to \{0,1\}\}_{n\in\mathbb{N}}$ such that any read-once parity branching program computing f_n has size $\Omega(n^2/\log^2 n)$.*

Here "explicit" means that there exists a polynomial-time Turing machine M that on input $x = (x_0, \ldots, x_{n-1})$ computes $f_n(x)$.

Note that in Definition 1.1 we didn't insist that the graph is layered. Jukna [Juk12] similarly does not require branching programs to be layered graphs. Indeed, when branching programs are used as a model of computation in a circuit complexity context (as opposed to using them to model space-bounded computation of Turing machines), requiring them to be layered imposes

a rather artificial constraint. Of course, any branching program can be made to be layered with a polynomial blow-up, so this wouldn't have mattered if we could prove super-polynomial lower bounds. In fact, if we assume that the graph is layered, we can prove a slightly better $\Omega(n^2)$ lower bound much more easily. We provide the details in Section 2.2.

We further remark that (again, as in [Juk12]), we measure the size of the program by the number of edges: since the in-degree of any vertex is unbounded, this is a natural complexity measure. This is another distinction that is only important insofar as the lower bounds we can prove are merely polynomial.

1.2 Technique

As mentioned above, we observe a natural connection between the problem of proving lower bounds for read-once $\oplus$-BPs and the problem of proving lower bounds for multilinear algebraic branching programs, a long-standing problem in algebraic complexity theory (see, for example, the recent works [CKSS24, FLSY26]).

A natural "algebrization" operation on parity branching programs computing a boolean function h results in a syntactic multilinear branching program, a well-studied model in algebraic complexity (see Section 2). Further, applying this operation on any read-once parity branching program would give an algebraic branching program computing the unique multilinear polynomial that agrees with h on $\mathbb{F}_2^n$.

To prove our lower bound, we resort to a result of Alon, Kumar and the author [AKV20] which proves such a lower bound for the model of syntactically multilinear *circuits* (improving an earlier result of [RSY08]). Circuits are stronger than branching programs, but when one deals with lower bounds that are merely super-linear (rather than super-polynomial) one has to be a bit careful when defining the model. However it turns out that the lower bound does apply to the algebraic branching programs that are obtained by converting $\oplus$-BPs to algebraic models. These models are defined in Section 2.

The last remaining ingredient then is to prove that the family of functions for which the lower bound of [AKV20] applies is explicit (in the sense of being in P). This does not follow immediately from the results of [AKV20] (as the definition of their polynomial involves a sum over a set of exponential size), and requires some work. We give a dynamic programming algorithm that computes this function. This algorithm appears in Section 3.

As a by-product, we also slightly tighten the results of [AKV20] and prove a lower bound for a polynomial in VP (the lower bound in [AKV20] was claimed for a polynomial in VNP). The details appear in Section 4.

2 Syntactically Multilinear ABPs and Read Once Parity Branching Programs

We start by defining the algebraic analog of read-once $\oplus$-BPs.

Definition 2.1. *A syntactically multilinear algebraic branching program (ABP) over $\mathbb{F}_2$ is a directed, acyclic multigraph with a source node s and a target node t. Every edge in the graph is labeled either by a constant in $\mathbb{F}_2$ or a linear function in some x_i. On every $s \to t$ path, every variable appears at most once. Each $s \to t$ path computes the product of the labels on the path, and the program computes the sum, over all $s \to t$ paths, of the polynomials computed by the paths. The* size *of the program is the number of edges.*

Note that unlike some common definitions in the literature, we didn't allow edges to be labeled by arbitrary linear functions in the variables, but rather only by a constant or a linear function in a single variable.

Similarly, an algebraic circuit C is called *syntactically multilinear* if every multiplication gate in C multiplies two variable-disjoint subcircuits. Circuits are stronger than ABPs:

Claim 2.2. *Suppose $f \in \mathbb{F}_2[x_1, \ldots, x_n]$ is computed by a syntactically multilinear ABP of size S. Then f is computed by a syntactically multilinear circuit of size $O(S)$.*

Proof. Simulate the ABP vertex by vertex. For a vertex v, the polynomial computed by v, denoted f_v, is defined to be the polynomial computed by the sub-ABP whose source is s and sink is v. By induction from s, for every vertex v in the ABP we add a sum gate v' to the circuit computing f_v. If v has incoming edges from vertices $u_1, \ldots, u_k$ with edge labels $\ell_1, \ldots, \ell_k$, the sum gate v' computes $\sum_{i=1}^{k} \ell_i f_{u_i}$, where the gates u_i' computing f_{u_i} have been already added to the circuit by induction, and ℓ_i is a linear function in a single variable and can be computed by a circuit of size $O(1)$.

For every edge in the ABP we need to add $O(1)$ edges to the circuit. Since the ABP is syntactically multilinear, every product gate multiplies a linear function in a variable x_i by a subcircuit C in which x_i doesn't appear, so the circuit is syntactically multilinear. □

We now construct a polynomial that requires syntactically multilinear circuits (and hence syntactically multilinear ABPs) of size $\Omega(n^2/\log^2 n)$.

In what follows, we associate $\mathbb{Z}_n = \mathbb{Z}/(n)$ with the set $\{0, 1, \ldots, n-1\}$ with addition modulo n. However, we sometimes think of the elements of $\mathbb{Z}_n$ as integers under the natural ordering.

Definition 2.3. *Let $\mathbb{F}$ be a field and n an even integer. Let $B \subseteq \mathbb{Z}_n$ be a subset of size $n/2$. Construct a bijection $\sigma_B : B \to \mathbb{Z}_n \setminus B$ in the following manner: think of the n elements of $\mathbb{Z}_n$ arranged on a cycle in a clockwise direction. Starting from 0 and going clockwise, pick the first element $j \in B$ such that the next element on the cycle, k, is not in B. Define $\sigma_B(j) = k$, erase j and k from the cycle and continue in that manner until all elements in B are assigned values. Let $f_B(x_0, \ldots, x_{n-1}) = \prod_{j \in B}(x_j + x_{\sigma_B(j)})$. Finally, define the following polynomial in $\mathbb{F}[x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}]$:*

$$f(x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}) = \sum_{\substack{B \subseteq \mathbb{Z}_n \\ |B| = n/2}} \prod_{j \in B} y_j \cdot f_B(x_0, \ldots, x_{n-1}).$$

This polynomial was constructed by Raz, Shpilka and Yehudayoff [RSY08], who proved a super-linear lower bound on the size of syntactically multilinear circuits computing it. This lower bound was improved in [AKV20]:

Theorem 2.4 ([RSY08, AKV20])**.** *Let*

$$f(x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}) \in \mathbb{F}[x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}]$$

be defined as above. Any syntactically multilinear circuit computing f has size $\Omega(n^2/\log^2 n)$.

Theorem 2.4 holds over any field, in particular over $\mathbb{F}_2$. We remark that the proofs in [RSY08, AKV20] don't use the special structure of the bijection σ_B outlined above, but rather only need σ_B to be some bijection from B to $\mathbb{Z}_n \setminus B$. This structure however will come in handy in Section 3.

The following corollary follows immediately from Claim 2.2.

Corollary 2.5. *Any syntactically multilinear ABP computing f has size $\Omega(n^2/\log^2 n)$.*

2.1 Connections between the Algebraic Model and the Boolean Model

For a polynomial $f \in \mathbb{F}_2[x_0, \ldots, x_{n-1}]$, let $f_{\text{bool}} : \{0,1\}^n \to \{0,1\}$ be the boolean function that f represents. It is clear that an *upper bound* on the ABP complexity of f gives an upper bound on the $\oplus$-BP complexity of f_{bool}, by "booleanizing" the ABP: replacing every label $1+x_i$ by $\neg x_i$ and treating the new graph as a boolean $\oplus$-BP gives a $\oplus$-BP that computes the same function as f on any inputs in $\{0,1\}^n$, and therefore correctly computes f_{bool}.

This observation shows that solving Jukna's [Juk12, Research Problem 16.14] would have major consequences in algebraic complexity: indeed, proving such a lower bound for f_{bool} would show a lower bound on the syntactically multilinear ABP size of f. Furthermore, since syntactically multilinear circuits can be simulated by formulas (and hence ABPs) with a quasi-polynomial blow-up [RY08], such a result would even imply an exponential lower bound on syntactically multilinear *circuits*.

We remark that Jukna's oblivious model corresponds to a model called read-once oblivious ABPs which was well-studied in algebraic complexity (see, e.g., [FS13]).

In the read-once setting, one could also deduce boolean complexity *lower bounds* from algebraic complexity lower bounds. The following claim is incredibly simple, but it is perhaps the key point behind our lower bound.

Claim 2.6. *Let $f \in \mathbb{F}_2[x_0, \ldots, x_{n-1}]$ be a multilinear polynomial. If f requires syntactically multilinear ABPs of size S, then f_{bool} requires read-once $\oplus$-BPs of size S.*

Proof. Consider any read-once $\oplus$-BP computing f_{bool} of size S'. "Algebrize" the branching program by replacing every label $\neg x_i$ by $1 + x_i$, and treating it as a syntactically multilinear ABP. This ABP computes a multilinear polynomial $g \in \mathbb{F}_2[x_0, \ldots, x_{n-1}]$ that agrees with f on $\mathbb{F}_2^n$, in the sense that for every x, $g(x) = f(x)$. Since f and g are both multilinear, $g = f$. Thus, we obtained a syntactically multilinear ABP computing f, which implies by our assumption that $S' \geq S$. □

Corollary 2.7. *Let f be as in Definition 2.3. Then f_{bool} requires read-once $\oplus$-BPs of size $\Omega(n^2/\log^2 n)$.*

Proof. Follows from Claim 2.6 and Corollary 2.5. □

Using nothing more than the definition in Definition 2.3, one could show that the function f_{bool} is in the class $\oplus\mathsf{P}$.

Claim 2.8. *$f_{bool} \in \oplus\mathsf{P}$.*

Proof. Consider a non-deterministic TM M that, on input $x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}$, guesses a subset $B \subseteq \mathbb{Z}_n$. If $|B| \neq n/2$, M rejects. Otherwise, M treats its input as elements in $\mathbb{F}_2^{2n}$, computes (using the notations of Definition 2.3)

$$\prod_{j \in B} y_j \cdot f_B$$

and accepts iff the result is 1. This last computation can be done in deterministic polynomial time. Then, on input $x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}$, M has an odd number of accepting paths iff $f_{\text{bool}}(x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}) = 1$. □

However, this is not entirely satisfying. Usually, in the context of circuit lower bounds, one would like to prove lower bounds for *explicit* functions, namely, functions in P (or NP). In Section 3 we prove that the function f_{bool} is actually explicit in that exact sense.

2.2 Quadratic Lower Bounds for Layered Branching Programs

Here we briefly remark that if we assume that the branching program is *layered*, we can obtain a truly quadratic $\Omega(n^2)$ lower bound (for a different function). Let

$$S_{n,d}(x_1,\ldots,x_n) = \sum_{\substack{B\subseteq[n]\\|B|=d}} \prod_{i\in B} x_i$$

denote the elementary symmetric polynomial of degree d. Chatterjee et al. [CKSV22] proved that any *layered* algebraic branching program computing $S_{n,n/10}$ over fields of characteristic 0 has size $\Omega(n^2)$. This lower bound does not assume multilinearity (and their model even allows the edge labels to be arbitrary affine functions in $x_1,\ldots,x_n$). A key ingredient in the proof is an upper bound on the dimension of the variety cut by the first order partial derivatives of $S_{n,d}$, proved by [MZ17, LMP19]. This upper bound was recently proved for any characteristic by Orzel [Orz25], which implies a lower bound for ABPs over $\mathbb{F}_2$ (which is necessary for us, as we consider algebraic computations over $\mathbb{F}_2$).

Consider then a read-once *layered* $\oplus$-BP of size S computing $(S_{n,n/10})_{\text{bool}}$. Applying the transformation in Claim 2.6, we obtain a layered algebraic branching program computing $S_{n,n/10}$, which implies, by [CKSV22, Orz25], that $S=\Omega(n^2)$. Note that Claim 2.6 uses the fact that the $\oplus$-BP is read-once. We cannot omit this condition, even though the lower bound proof of [CKSV22, Orz25] does not require the ABP to be multilinear.

Finally, note that $h := (S_{n,d})_{\text{bool}}$ is obviously an explicit function for any d, since for every $x\in\{0,1\}^n$, $h(x)=\binom{|x|}{d} \bmod 2$, where $|x|$ denotes the Hamming weight of x.

Chatterjee et al. [CKSV22] also proved lower bounds for *unlayered* branching programs, but these lower bounds are much weaker and not helpful for us in this context.

3 A Polynomial-Time Dynamic Programming Algorithm

We now present a polynomial-time algorithm for computing f_{bool}, where f is as defined in Definition 2.3. To that end, we adopt a more combinatorial view of what f_{bool} actually computes.

Consider again the elements of $\mathbb{Z}_n$ on a cycle and the process of constructing σ_B in Definition 2.3. We depict the action of matching j to $\sigma_B(j)$ as drawing a directed chord between j and $\sigma_B(j)$, labeled by y_j. We shall soon prove that these chords are always *non-crossing*. An example is depicted in Fig. 1

Given $x=(x_0,\ldots,x_{n-1})\in\{0,1\}^n$ we think of x as assigning bits on the n elements of $\mathbb{Z}_n$. A set B, along with its non-crossing matching σ_B, is called *x-valid* if for every $j\in B$, $x_j\neq x_{\sigma_B(j)}$. Similarly, given $y\in\{0,1\}^n$, we say that B is *y-eligible* if for all $j\in B$, y_j=1.

Note that since the additions and multiplications in Definition 2.3 are modulo 2, for every input $x=(x_0,\ldots,x_{n-1})$, we have that $f_B(x)=1$ if and only if B is x-valid. It follows that given an input $(x,y)\in\{0,1\}^n\times\{0,1\}^n$, the function f_{bool} counts, modulo 2, the number of x-valid and y-eligible sets B. We will show that one can in fact count this number exactly in polynomial time (and therefore trivially compute its parity).

Suppose that instead of a *cycle* of length n we were to construct a non-crossing matching on an *interval* of length n in a similar fashion (one may think of this process as matching opening parentheses '(' with closing parentheses ')' in a well-matched parentheses sequence, but of course, on an interval not every subset B of $n/2$ opening parentheses corresponds to a well-matched sequence). Ignoring the y part for the time being, this setting naturally lends itself to a dynamic programming algorithm: we construct a table $M_{i,j}$ in which the (i,j)-th cell counts the number of x-valid matchings in the subinterval $[i,j]$, with our eventual goal being to compute $M_{0,n-1}$. To compute $M_{i,j}$, we use the fact that i must be matched to some element in $[i+1,j]$, which

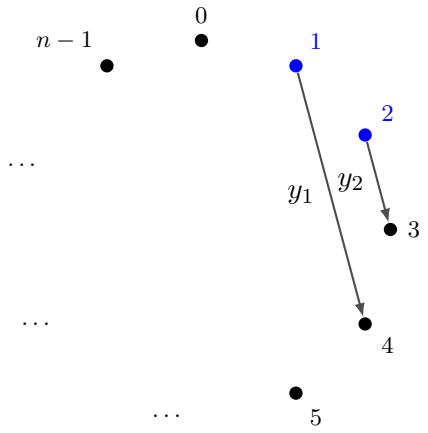

Figure 1: Cycle with chords. Blue nodes are in B and black nodes are not in B. The matching σ_B maps 2 to 3 by an edge labeled y_2 and then 1 to 4 by an edge labeled y_1.

gives the recursive formula

$$M_{i,j} = \sum_{k \in [i+1,j], x_k \neq x_i} M_{i+1,k-1} \cdot M_{k+1,j}.$$

(The "$x_k \neq x_i$" condition makes sure we only count x-valid matchings). The base cases for this induction are empty intervals whose value is 1.

Our dynamic programming algorithm is inspired by this observation, but the fact that we are working with a cycle and not an interval means that some of the matchings can "wrap around" and are not accounted for by the formula above.

To solve this issue, we instead perform the count slightly differently. Consider again the set B along with a matching σ_B. Let ℓ be the smallest element $j \in \mathbb{Z}_n$ such that $\sigma_B(j) < j$ when considered as integers (if there's no such element, set $\ell = 0$). We say that ℓ is the *leader* of the matching.

We now state and prove a useful combinatorial lemma.

Lemma 3.1. *Let $B \subseteq \mathbb{Z}_n$ be a subset of size $n/2$ and consider the matching σ_B as constructed in Definition 2.3. Let ℓ be the leader of the matching (as defined above). "Cut" the cycle at ℓ so that we get the interval*

$$\ell, \ell+1, \ell+2, \dots, n-1, 0, 1, 2, \dots, \ell-1$$

For every $j \in B$, draw a directed edge between j and $\sigma_B(j)$ on this interval (an illustration of this operation appears in Fig. 2). Then these edges are non-crossing, not wrapping around, and they all go from left to right.

Proof. If there's no element $j \in \mathbb{Z}_n$ such that $\sigma_B(j) < j$ then $\ell = 0$. In this case, the interval equals $0, \dots, n-1$, and the fact that the edges all go from left to right is rather obvious, as $\sigma_B(j) > j$ for all j. The proof of the non-crossing property is by induction on the construction of σ_B. We claim that at each stage, when we match j to $\sigma_B(j)$, all elements in $[j+1, \sigma_B(j)-1]$ have already been matched. This is definitely true in the first stage, as we match j to $j+1$ so that interval is empty. At any later step, we similarly match j to its neighbor on the cycle k. If k is a neighbor of j it means that all elements in $[j+1, k-1]$ were already deleted and hence matched before.

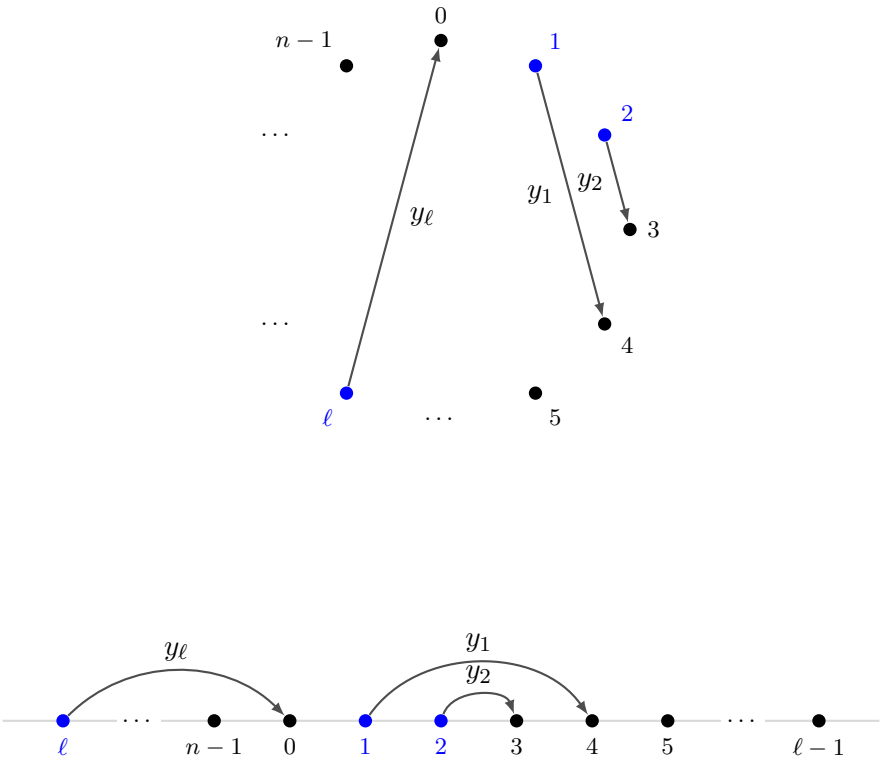

Figure 2: "Cutting" a cycle at the leader ℓ and drawing the edges on an interval.

Suppose now $\ell > 0$ is a leader such that $\sigma_B(\ell) < \ell$, and order the elements as

$$\ell, \ell+1, \ell+2, \ldots, n-1, 0, 1, 2, \ldots, \sigma_B(\ell), \ldots, \ell-1$$

By definition there is a directed edge from ℓ to $\sigma_B(\ell)$. As before, since ℓ was connected to $\sigma_B(\ell)$ it means that all elements on the arc $[\ell+1, \sigma_B(\ell)-1]$ of the cycle were already matched to one another and erased from the cycle. Hence the elements in each of the intervals $[\ell+1, \sigma_B(\ell)-1]$ and $[\sigma_B(\ell)+1, \ell-1]$ are matched among themselves and the edge from ℓ to $\sigma_B(\ell)$ doesn't intersect any other edge, and no edge wraps around. With an identical argument we can argue that in each interval there are no intersecting edges.

To prove that each edge goes from left to right, consider a matched pair $j \in B$ and $\sigma_B(j) \notin B$. If $j = \ell$ this was already established. If $j \in [0, \ell-1]$, then by the definition of a leader we must have $\sigma_B(j) > j$ and clearly in the interval as ordered above the edge goes from left to right. The final case to consider is thus $j \in [\ell+1, n-1]$: suppose that $\sigma_B(j)$ resides to the left of j in the ordering above, both lying in the interval between ℓ and $\sigma_B(\ell)$. In the construction of σ_B, we try to match an element with its neighboring element on the cycle, ordered clockwise. We could have only matched ℓ with $\sigma_B(\ell)$ if all the elements

$$\ell+1, \ell+2, \ldots, n-1, 0, 1, \ldots, \sigma_B(\ell)-1 \tag{1}$$

were already matched among themselves, which means that when j was matched to $\sigma_B(j)$, ℓ and $\sigma_B(\ell)$ were not erased yet. Since we match adjacent elements directed clockwise, we must have $\sigma_B(j)$ appearing to the right of j in the subinterval (1), as otherwise j could not be adjacent to $\sigma_B(j)$ (since ℓ was not yet erased). □

Conversely, given any such cyclic shift of an interval with leader ℓ and labeled matched edges, we can uniquely recover the set B and the mapping σ_B using the direction of the edges (recall that an edge directed from j to k implies that $j \in B$ and $k \notin B$).

Consider now an *interval* (rather than an arc) $[i, j]$, $i \leq j$. We say that a set B with a non-crossing matching σ_B is *external* to $[i, j]$ if ℓ is not in the interval $[i, j]$. Let $\mathsf{Ext}_{i,j}$ denote the number of possible ways to match the elements of $[i, j]$, using x-valid and y-eligible *external* sets B.

Similarly, B is *internal* to $[i, j]$ if ℓ is inside the interval $[i, j]$, and let $\mathsf{Int}_{i,j}$ denote the number of possible ways to match the elements of $[i, j]$, using x-valid and y-eligible *internal* sets B.

Since every B is internal to $[0, n-1]$, we are interested in computing $\mathsf{Int}_{0,n-1}$.

We are now ready to prove the main theorem of this section.

Theorem 3.2. *There exists a polynomial-time algorithm that, given $x, y \in \{0,1\}^n \times \{0,1\}^n$, computes the number of x-valid and y-eligible sets B.*

Proof of Theorem 3.2. We compute $\mathsf{Ext}_{i,j}$ and $\mathsf{Int}_{i,j}$ by induction on the length of the interval. The length needs to be even for such matchings to exist, that is, if $i \leq j$ and the length $(j - i + 1)$ is an odd number then $\mathsf{Ext}_{i,j} = \mathsf{Int}_{i,j} = 0$. Further, if the interval is empty, that is $j < i$, $\mathsf{Ext}_{i,j} = \mathsf{Int}_{i,j} = 1$.

For larger lengths, our first claim is the following:

Claim 3.3.

$$\mathsf{Ext}_{i,j} = \sum_{k \in [i+1,j], x_i \neq x_k} y_i \cdot \mathsf{Ext}_{i+1,k-1} \cdot \mathsf{Ext}_{k+1,j}. \tag{2}$$

Proof of Claim 3.3. Indeed, the equation goes over all possible ways to match the element i with an element $k \in [i, j]$. Since we're only counting x-valid matchings, we only need to consider indices k such that $x_i \neq x_k$. Since the leader ℓ is not in $[i, j]$, it is not in $[i + 1, k - 1]$ nor in $[k + 1, j]$, so we multiply the relevant number of external matchings for these subintervals. Further, we claim that since the leader ℓ is not in $[i, j]$ it must be that $i \in B$ and $k \notin B$: if $\ell < i$, then by cutting the cycle at ℓ we obtain the interval

$$\ell, \ell + 1, \ldots, i, \ldots, k, \ldots, j \ldots, n - 1, 0, 1, \ldots, \ell - 1.$$

By Lemma 3.1, since the directed edges go from left to right, we see that if i is matched to k we must have $i \in B, k \notin B$.

On the other hand, if $\ell > j$, then since ℓ is the smallest element with $\sigma_B(\ell) < \ell$, and $i < k < j < \ell$, we must have $k = \sigma_B(i)$ and thus $i \in B$ and $k \notin B$.

Since we established that $i \in B$ and $k \notin B$ in both cases, we multiply by y_i to only count y-eligible matchings. □

Now consider internal matchings.

Claim 3.4.

$$\mathsf{Int}_{i,j} = \sum_{k \in [i+1,j], x_i \neq x_k} \left(y_k \cdot \mathsf{Int}_{i+1,k-1} \cdot \mathsf{Ext}_{k+1,j} + y_i \cdot \mathsf{Ext}_{i+1,k-1} \cdot \mathsf{Int}_{k+1,j} \right). \tag{3}$$

Proof of Claim 3.4. Here we count matchings in which the leader ℓ is inside $[i, j]$. We again go over all possible elements k that can be matched to i. We have either $\ell \in [i, k]$ or $\ell \in [k + 1, j]$.

In the latter case ($\ell \in [k + 1, j]$), by definition of ℓ, and since $k < \ell$, the directed edge must go from i to k and therefore $i \in B, k \notin B$ (as otherwise k would be a smaller element than ℓ with $\sigma_B(k) < k$, which contradicts the definition of ℓ). We thus multiply the number of external matchings on $[i + 1, k - 1]$ (since the leader is not in that interval) by the number of internal matchings on $[k + 1, j]$, going over all k such that $x_k \neq x_i$ (to only count x-valid matchings)

and multiplying by y_i (to only count y-eligible matchings). This accounts for the second term in (3).

We are left with the case $\ell \in [i,k]$. In this case, when we cut the cycle at position ℓ,

$$\ell, \ell+1, \ldots, k, \ldots, j, \ldots, n-1, 0, 1, \ldots, i, \ldots, \sigma_B(\ell), \ldots, \ell-1$$

By Lemma 3.1 we see that the edge must be directed from k to i, that is, $k \in B$ and $i = \sigma_B(k) \notin B$. We thus multiply the number of internal matchings on $[i+1, k-1]$ by the number of external matchings on $[k+1, j]$, going over all k such that $x_k \neq x_i$ (to only count x-valid matchings) and multiplying by y_k (to only count y-eligible matchings). This accounts for the first term in (3). □

Claim 3.3 and Claim 3.4 now establish Theorem 3.2. As noted above, in order to compute $\mathsf{Int}_{0,n-1}$ we recursively compute $\mathsf{Int}_{i,j}$ and $\mathsf{Ext}_{i,j}$ for all $i < j$, by induction on the length of the interval. There are $O(n^2)$ quantities to compute, and using (2) and (3), each can be computed in time $O(n)$. □

4 Algebraic Circuits Lower Bound for a Polynomial in VP

Our proof from Section 3 also implies that the polynomial f from Definition 2.3 is in VP, the class of polynomial families of degree $\mathsf{poly}(n)$ and circuits of size $\mathsf{poly}(n)$ (in [AKV20], it is only claimed to be in VNP). This shows that the lower bound of [AKV20] also holds for a polynomial in VP. We remark that the technical condition that Alon et al. [AKV20] need f to satisfy is that its coefficient matrix is full rank under any partition of the variables. This technique was introduced by Raz [Raz09] and was later also used in [Raz06, RY08], to name only a few examples. A more systematic study of this technique appears in [FLSY26]. We do not go into details here and refer to any of these papers for precise definitions. We call such a polynomial a *full rank polynomial*.

A full rank polynomial in VP was already constructed in [RY08]. For technical reasons, however, in the proof one needs to consider such a polynomial $f(x,y) \in \mathbb{F}_2[x,y]$ as a polynomial in x over the field $\mathbb{F}(y)$ (here x, y are vectors of variables), and then the rank is computed over $\mathbb{F}(y)$. In the construction of Raz and Yehudayoff [RY08], the number of variables in y is $O(n^3)$, whereas the lower bound of [AKV20] is nearly-quadratic in the number of variables in x. Therefore, if the complexity is measured as a function of the total number of variables, the lower bound is meaningless for the polynomial of [RY08] (in f from Definition 2.3 the number of variables in y is n, so this problem doesn't arise). When $\mathbb{F}$ is large enough, one can take the construction of Raz and Yehudayoff [RY08] and plug in random values to the y variables. With high probability, after this fixing, one obtains a full-rank n-variate polynomial over $\mathbb{F}$. This construction, however, is not explicit (and requires large fields). For further discussion on this topic see Section 4 of [AKV20].

Fortunately, this somewhat annoying issue is no longer an issue, since we can prove:

Theorem 4.1. *Let f be as in Definition 2.3. Then f has a circuit of size $O(n^3)$ (and in particular, $f \in \mathsf{VP}$).*

Proof. We construct a circuit following the proof of Theorem 3.2, using equations similar to (2) and (3). For every $i \leq j$ such that $j - i + 1$ is even we add two gates $E_{i,j}$, $I_{i,j}$ and connect them as follows:

$$E_{i,j} = \sum_{k \in [i+1,j]} y_i \cdot (x_i + x_k) \cdot E_{i+1,k-1} \cdot E_{k+1,j}$$

$$I_{i,j} = \sum_{k \in [i+1,j]} \left(y_k \cdot (x_i + x_k) \cdot I_{i+1,k-1} \cdot E_{k+1,j} + y_i \cdot (x_i + x_k) \cdot E_{i+1,k-1} \cdot I_{k+1,j} \right)$$

(where each gate $E_{i',i''}$ is understood to be the constant 1 if $i'' < i'$, and similarly for $I_{i',i''}$). The output of the circuit is the gate $I_{0,n-1}$.

The circuit C is multilinear: it follows by induction on $j - i + 1$ that $E_{i,j}$ and $I_{i,j}$ are multilinear since the subcircuits rooted at them are only connected to x and y variables with indices in $[i, j]$.

One can prove by induction that C computes f. A different way to see it is by directly reducing to Theorem 3.2. Since the circuit C agrees with f_{bool} functionally on $\{0,1\}^n \times \{0,1\}^n$, and since it is multilinear, it must compute the polynomial f.

The circuit C has size $O(n^3)$: the fan-in of each of the $O(n^2)$ gates $I_{i,j}$ and $E_{i,j}$ is $O(n)$, for a total of $O(n^3)$ edges. One can then convert this circuit to a bounded fan-in circuit with $O(n^3)$ gates. □

Note that the polynomial in Definition 2.3 is defined over any field and Theorem 4.1 is true over any field.

Theorem 4.1 shows a barrier for the technique of analyzing the rank of the coefficient matrix under various partitions: it can't prove lower bounds beyond $\Omega(n^3)$ (strictly speaking, this also follows from the randomized construction mentioned above using the polynomial of Raz and Yehudayoff [RY08]. The *existence* of a circuit of size $O(n^3)$ computing a full-rank polynomial, even non-explicitly, is enough to prove the barrier result).

5 Open Problems

One could hope, of course, to solve [Juk12, Research Problem 16.14] completely and prove super-polynomial lower bounds for read-once parity branching programs. Such a result would follow from super-polynomial lower bounds for syntactically multilinear algebraic branching programs. A study of the limitations of current techniques for proving such lower bounds was recently initiated by Fabris et al. [FLSY26]. As a first step, we propose proving a cubic $\Omega(n^3)$ lower bound, perhaps by proving such a lower bound on syntactically multilinear algebraic circuits: that would prove that the construction in Theorem 4.1 is optimal, but it's worth mentioning that we have no strong reasons to believe that it is indeed optimal, or that there isn't another full-rank polynomial with a circuit of size $O(n^2)$ (by the results of [AKV20], such a construction *would* be optimal, up to logarithmic factors).

More generally, for many boolean models of computation, the best lower bounds known are proved using Nechiporuk's [Nec66] method. It is interesting to try and find more cases in which stronger lower bounds can be proved using various methods, in particular using reductions to lower bounds in algebraic circuit complexity.

References

[AKV20] Noga Alon, Mrinal Kumar, and Ben Lee Volk. Unbalancing Sets and An Almost Quadratic Lower Bound for Syntactically Multilinear Arithmetic Circuits. *Comb.*, 40(2):149–178, 2020.

[BHST87] László Babai, Péter Hajnal, Endre Szemerédi, and György Turán. A Lower Bound for Read-Once-Only Branching Programs. *J. Comput. Syst. Sci.*, 35(2):153–162, 1987.

[BHW03] Henrik Brosenne, Matthias Homeister, and Stephan Waack. Lower Bounds for General Graph-Driven Read-Once Parity Branching Programs. In *Mathematical Foundations of Computer Science (MFCS)*, volume 2747 of *Lecture Notes in Computer Science*, pages 290–299. Springer, 2003.

[BRS93] Allan Borodin, Alexander A. Razborov, and Roman Smolensky. On Lower Bounds for Read-k-Times Branching Programs. *Computational Complexity*, 3:1–18, 1993.

[BS83] Walter Baur and Volker Strassen. The Complexity of Partial Derivatives. *Theor. Comput. Sci.*, 22:317–330, 1983.

[CHMY24] Mahdi Cheraghchi, Shuichi Hirahara, Dimitrios Myrisiotis, and Yuichi Yoshida. One-Tape Turing Machine and Branching Program Lower Bounds for MCSP. *Theory Comput. Syst.*, 68(4):868–899, 2024.

[CKSS24] Prerona Chatterjee, Deepanshu Kush, Shubhangi Saraf, and Amir Shpilka. Lower Bounds for Set-Multilinear Branching Programs. In *39th Computational Complexity Conference (CCC)*, volume 300 of *LIPIcs*, pages 20:1–20:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[CKSV22] Prerona Chatterjee, Mrinal Kumar, Adrian She, and Ben Lee Volk. Quadratic Lower Bounds for Algebraic Branching Programs and Formulas. *Comput. Complex.*, 31(2):8, 2022.

[FKS16] Michael A. Forbes, Mrinal Kumar, and Ramprasad Saptharishi. Functional Lower Bounds for Arithmetic Circuits and Connections to Boolean Circuit Complexity. In *31st Conference on Computational Complexity (CCC)*, volume 50 of *LIPIcs*, pages 33:1–33:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[FLSY26] Théo Borém Fabris, Nutan Limaye, Srikanth Srinivasan, and Amir Yehudayoff. Multilinear Algebraic Branching Programs and the Min-Partition Rank Method. *Electron. Colloquium Comput. Complex.*, TR26, 2026. Pre-print available at `arXiv:TR26-001`.

[For24] Michael A. Forbes. Low-Depth Algebraic Circuit Lower Bounds over Any Field. In *39th Computational Complexity Conference (CCC)*, volume 300 of *LIPIcs*, pages 31:1–31:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[FS13] Michael A. Forbes and Amir Shpilka. Quasipolynomial-Time Identity Testing of Non-commutative and Read-Once Oblivious Algebraic Branching Programs. In *54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 243–252. IEEE Computer Society, 2013.

[FSTW21] Michael A. Forbes, Amir Shpilka, Iddo Tzameret, and Avi Wigderson. Proof Complexity Lower Bounds from Algebraic Circuit Complexity. *Theory Comput.*, 17:1–88, 2021.

[GR00] Dima Grigoriev and Alexander A. Razborov. Exponential Lower Bounds for Depth 3 Arithmetic Circuits in Algebras of Functions over Finite Fields. *Appl. Algebra Eng. Commun. Comput.*, 10(6):465–487, 2000.

[HLT24] Tuomas Hakoniemi, Nutan Limaye, and Iddo Tzameret. Functional Lower Bounds in Algebraic Proofs: Symmetry, Lifting, and Barriers. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1396–1404. ACM, 2024.

[Hom06] Matthias Homeister. Lower bounds for restricted read-once parity branching programs. *Theor. Comput. Sci.*, 359(1-3):1–14, 2006.

[JS82] Mark Jerrum and Marc Snir. Some Exact Complexity Results for Straight-Line Computations over Semirings. *Journal of the ACM*, 29(3):874–897, 1982.

[Juk12] Stasys Jukna. *Boolean Function Complexity - Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer, 2012.

[Kal85] Kyriakos Kalorkoti. A Lower Bound for the Formula Size of Rational Functions. *SIAM J. Comput.*, 14(3):678–687, 1985.

[KW93] Mauricio Karchmer and Avi Wigderson. On Span Programs. In *Proceedings of the Eighth Annual Structure in Complexity Theory Conference*, pages 102–111. IEEE Computer Society, 1993.

[LMP19] Nutan Limaye, Kunal Mittal, and Mukesh Pareek. Homogeneous ABP complexity of elementary symmetric polynomial, 2019.

[LST25] Nutan Limaye, Srikanth Srinivasan, and Sébastien Tavenas. Superpolynomial Lower Bounds Against Low-Depth Algebraic Circuits. *J. ACM*, 72(4):26:1–26:35, 2025.

[MZ17] Izaak Meckler and Gjergji Zaimi. Singular locus of zero locus of elementary symmetric polynomials, 2017.

[Nec66] Eduard Ivanovich Nechiporuk. On a Boolean function. *Dokl. Akad. Nauk SSSR*, 169:765–766, 1966.

[Nis91] Noam Nisan. Lower Bounds for Non-Commutative Computation (Extended Abstract). In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 410–418. ACM, 1991.

[Oko91] E.A. Okolnishnikova. Lower bounds on the complexity of realization of characteristic functions of binary codes by branching programs. *Metody Diskretnogo Analiza*, 51:61–83, 1991.

[Orz25] Ian Orzel. Computing the Elementary Symmetric Polynomials in Positive Characteristics. *Electron. Colloquium Comput. Complex.*, TR25, 2025. Pre-print available at `arXiv:TR25-128`.

[Raz06] Ran Raz. Separation of Multilinear Circuit and Formula Size. *Theory Comput.*, 2(6):121–135, 2006.

[Raz09] Ran Raz. Multi-linear formulas for permanent and determinant are of super-polynomial size. *J. ACM*, 56(2):8:1–8:17, 2009.

[RSY08] Ran Raz, Amir Shpilka, and Amir Yehudayoff. A Lower Bound for the Size of Syntactically Multilinear Arithmetic Circuits. *SIAM J. Comput.*, 38(4):1624–1647, 2008.

[RY08] Ran Raz and Amir Yehudayoff. Balancing Syntactically Multilinear Arithmetic Circuits. *Comput. Complex.*, 17(4):515–535, 2008.

[RY09] Ran Raz and Amir Yehudayoff. Lower Bounds and Separations for Constant Depth Multilinear Circuits. *Comput. Complex.*, 18(2):171–207, 2009.

[Sap15] Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity. Github survey, 2015.

[SS05] Petr Savický and Detlef Sieling. A hierarchy result for read-once branching programs with restricted parity nondeterminism. *Theor. Comput. Sci.*, 340(3):594–605, 2005.

[Str73] Volker Strassen. Die Berechnungskomplexität Von Elementarsymmetrischen Funktionen Und Von Interpolationskoeffizienten. *Numerische Mathematik*, 20(3):238–251, June 1973.

[SY10] Amir Shpilka and Amir Yehudayoff. Arithmetic Circuits: A survey of recent results and open questions. *Found. Trends Theor. Comput. Sci.*, 5(3-4):207–388, 2010.

[Tha98] Jayram S. Thathachar. On Separating the Read-k-Times Branching Program Hierarchy. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing (STOC)*, pages 653–662. ACM, 1998.

[Weg88] Ingo Wegener. On the complexity of branching programs and decision trees for clique functions. *J. ACM*, 35(2):461–471, 1988.

[Zák84] Stanislav Zák. An Exponential Lower Bound for One-Time-Only Branching Programs. In *Mathematical Foundations of Computer Science (MFCS)*, volume 176 of *Lecture Notes in Computer Science*, pages 562–566. Springer, 1984.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il