# Monomials in Read-Once/Twice Formulas and Branching Programs*

Meena Mahajan[1], B V Raghavendra Rao[2], and Karteek Sreenivasaiah[1]

[1] Institute of Mathematical Sciences, Chennai, India. {meena,karteek}@imsc.res.in
[2] University of Saarland, Saarbrücken, Germany. bvrr@cs.uni-sb.de

**Abstract.** We study three computational problems on arithmetic circuits. Given an arithmetic circuit $C$, 1) ZMC: test if a given monomial in $C$ has zero coefficient, 2) MonCount: compute the number of monomials in $C$, and 3) MLIN: test if $C$ computes a multilinear polynomial. These problems were introduced by Fournier, Malod and Mengel [STACS 2012], and shown to characterize various levels of the counting hierarchy (CH). We address the above problems on read-restricted arithmetic circuits and branching programs. We prove several complexity characterizations for the above problems on these restricted classes of arithmetic circuits. Along the way, we also obtain a simple non-black box algorithm for the arithmetic circuit identity testing problem (ACIT) on read-twice formulas. To the best of our knowledge, this is the first deterministic polynomial time algorithm for ACIT on read-twice formulas.

## 1 Introduction

A fundamental question one can ask concerning a given arithmetic circuit is: does the circuit compute the identically zero polynomial? This is the well-known problem Arithmetic Circuit Identity Testing ACIT, that has spurred an enormous amount of research in the last two decades. A complete derandomization of black-box ACIT even for the case of depth four arithmetic circuits implies circuit lower bounds [12, 1].

A recent paper by Fournier, Malod and Mengel [10] studies two related, and apparently harder, problems: (1) MonCount: compute the number of monomials in the polynomial computed by a given circuit, and (2) ZMC: Decide whether a given monomial has zero coefficient in the polynomial computed by a given circuit. The circuits are allowed to use only the constants $\{-1, 0, 1\}$. (Restricting the allowed constants seems necessary to make the connection to uniform complexity classes.) The authors show that these problems are complete for levels of the counting hierarchy CH (the hierarchy based on the complexity classes PP and $C_=P$). They also show that if the circuits compute multilinear polynomials, then these problems become easier (equivalent to PP and ACIT respectively), and that multilinearity checking itself is equivalent to ACIT. All these results from [10] are in the non-black-box model, where the circuit is given explicitly in

* partially supported by Indo-German Max Planck Center (IMPECS)

the input. For ZMC, their results improve those of [13], where a weaker upper bound (albeit still in CH) is shown.

In this paper, we investigate the complexity of these problems when the circuits are severely restricted, and obtain bounds corresponding to complexity classes inside P. A very natural well-studied restriction is when the circuit is a formula; evaluation of such formulas on Boolean-valued inputs is complete for the arithmetic class $\mathsf{GapNC}^1$. Even in these cases, ACIT remains notoriously hard, and so we consider further restrictions. The simplest kind of formulas are read-once formulas ROFs: every variable appears at most once. Deterministic polynomial-time algorithms for ACIT on such formulas are trivial, and are known even on some generalizations of these, [14, 15, 4]. We show that MonCount and ZMC for this class are in the $\mathsf{GapNC}^1$ hierarchy and in logspace respectively (Theorem 1 and Theorem 5). It is straightforward to see that ZMC for ROFs is hard for $\mathsf{C_=NC}^1$, so this is almost tight.

Another equally natural and well-studied restriction is when the circuit is an algebraic branching program BP with edges labeled by the allowed constants or by variables. Evaluation of BPs on Boolean-valued inputs is complete for the arithmetic class GapL, the logspace analogue of the class GapP. Three restrictions, in order of increasing generality, are: (1) occur-once BPs, or OBPs, where each variable appears at most once anywhere in the BP (also called *global read-once* BPs), these are known to subsume ROFs, (2) read-once BPs, or RBPs, where no path has two occurrences of the same variable, and (3) multilinear BPs, or MBPs, where the polynomial computed at every node is multilinear. Again, deterministic algorithms are known for ACIT on OBPs, [11]. We show that MonCount for OBPs is in the GapL hierarchy (Theorem 3), while ZMC for OBPs and even MBPs is complete for the complexity class $\mathsf{C_=L}$ (Theorem 4).

A related problem explored in [10] as a tool to solving MonCount is that of checking, given a circuit $C$ and monomial $m$, whether $C$ computes any monomial that extends $m$. Denote this problem ExistExtMon. Though our algorithms for MonCount do not need this subroutine, we also show that for OBPs (and hence for ROFs), ExistExtMon lies in the GapL hierarchy (Theorem 6).

Finally, we consider how these results can be pushed beyond the read-once case. The read-once restriction guarantees that the polynomials computed at any gate of the circuit (ROF or RBP) are multilinear, and this property is crucially used in our algorithms. In fact, it is crucially used in many settings. For instance, if the formula is read-$k$ for some constant $k$, **and** is also multilinear, then ACIT is in P [4]. If the input circuit is not read-once, can we efficiently decide if this algorithm is applicable? That is, can we efficiently check whether the multilinearity property holds? For general circuits, this is equivalent to ACIT, as shown in [10]. We show that for read-twice formulas, we can check multilinearity and test for identically zero polynomials in deterministic polynomial time (Theorem 8). To the best of our knowledge, this gives the first (non-black-box) polynomial time algorithm for ACIT on read-twice formulas.

Owing to space constraints, some proofs appear in the Appendix.

## 2 Preliminaries

**Circuits, formulas, branching programs.** Let $X = \{x_1, \ldots, x_n\}$ be a set of variables. An *arithmetic circuit* $C$ over a ring $R$ is a directed acyclic graph with internal nodes labeled $+$ or $\times$ and leaves labeled from $X \cup R$. Every node has in-degree zero or two, and there is at least one node of out-degree zero, called the output gate. Unless otherwise stated, we consider $R$ to be the ring of integers $\mathbb{Z}$, and we allow only the constants $\{-1, 0, 1\}$ in the circuits. An *arithmetic formula* $F$ is an arithmetic circuit where fan-out for every gate is at most one.

Every node in $C$ computes a polynomial in $R[x_1, \ldots, x_n]$ in a natural way. For gate $g$ in $C$, we denote by $p_g$ the polynomial computed at $g$. We denote by $p_C$ the polynomial $p_r$, where $r$ is the output gate of $C$. We define the set

$$\mathsf{var_g} = \{\mathsf{x_i} \mid \text{some descendant of } g \text{ is a leaf labelled } x_i\}.$$

A *read-once* arithmetic formula (ROF for short) is an arithmetic formula where each variable occurs at most once as a label. More generally, in a *read-k* arithmetic formula a variable occurs at most $k$ times as a label.

An algebraic branching program (ABP) over $R$ is an undirected acyclic graph $B$ with edges labeled from $X \cup R$, and with two designated nodes, $s$ with zero in-degree, and $t$ with zero out-degree. For any directed path $\rho$ in $B$, define

$$\mathsf{weight}(\rho) = \prod_{\mathsf{e}:\ \text{an edge in } \rho} \mathsf{label}(\mathsf{e}).$$

A pair of nodes $u$, $v$ in $B$ computes a polynomial defined as follows:

$$p_B(u, v) = \sum_{\rho:\ \rho \text{ is a } u \rightsquigarrow v \text{ path in } B} \mathsf{weight}(\rho).$$

The ABP $B$ computes the polynomial $p_B \stackrel{\triangle}{=} p_B(s, t)$. We drop the subscript $B$ from the above when clear from context.

We consider the following restrictions of ABPs in increasing order of generality: (1) occur-once ABPs, or OBPs, where each variable appears at most once anywhere in the ABP (such BPs generalize ROFs), (2) read-once ABPs, or RBPs, where no path has two occurrences of the same variable, and (3) multilinear BPs, or MBPs, where the polynomial computed at every node is multilinear.

**Complexity Classes.** For standard complexity classes, the reader is referred to [5]. We define some of the non-standard complexity classes that are used in the paper. Let $f = (f_n)_{n \geq 0}$ be a family of integer valued functions $f_n : \{0,1\}^n \to \mathbb{Z}$. $f$ is in the complexity class $\mathsf{GapL}$ exactly when there is some nondeterministic logspace machine $M$ such that for every $x$, $f(x)$ equals the number of accepting paths of $M$ on $x$ minus the the number of rejecting paths of $M$ on $x$. $\mathsf{C_=L}$ is the class of languages $L$ such that for some $f \in \mathsf{GapL}$, for every $x$, $x \in L \Leftrightarrow f(x) = 0$. The $\mathsf{GapL}$ hierarchy is built over $\mathsf{C_=L}$ languages or bit access to $\mathsf{GapL}$ functions,

with a deterministic logspace machine at the base, and is known to be contained in $\mathsf{NC}^2$. (See [3, 2] for more details.)

$\mathsf{GapNC}$ denotes the class of families of functions $(f_n)_{(n\geq 0)}$, $f_n : \{0,1\} \to \mathbb{Z}$, where $f_n$ can be computed by a uniform polynomial size log depth arithmetic circuit. This equals the class of functions computed by uniform polynomial-sized arithmetic formulas ([7]). $\mathsf{C}_{=}\mathsf{NC}^1$ is the class of languages $L$ such that for some $\mathsf{GapNC}$ function family $(f_n)_{n\geq 0}$, and for every $x$, $x \in L \iff f_{|x|}(x) = 0$. The $\mathsf{GapNC}$ hierarchy comprises of languages accepted by polynomial-size constant depth unbounded fanin circuits ($\mathsf{AC}^0$) with oracle access to bits of $\mathsf{GapNC}$ functions, and is known to be contained in $\mathsf{DLOG}$. (See [8, 9] for more details.)

**Miscellaneous Notation.** A monomial is represented by the sequence of degrees of the variables. For any set $S \subseteq [n]$, we denote by $m_S$ the multilinear monomial $\prod_{i\in S} x_i$. For a monomial $m$ and polynomial $p$, $\mathsf{coeff}(p, m)$ denotes the coefficient of $m$ in $p$.

[statement $S$] is a Boolean 0-1 valued predicate that is 1 exactly when the statement $S$ is true.

**Prolems Considered.** We now describe the computational problems considered in this paper.

*Problem 1 (*$\mathsf{MonCount}$*).* **Input**: An arithmetic circuit $C$ over $\mathbb{Z}$.
**Output**: The number of monomials in the polynomial computed by $C$.

*Problem 2 (*$\mathsf{MLIN}$*).* **Input**: An arithmetic formula $F$ over $\mathbb{Z}$.
**Output**: Test if the polynomial $p_F$ is multilinear or not.

*Problem 3 (*$\mathsf{ZMC}$*).* **Input**: An arithmetic circuit $C$ over $\mathbb{Z}$, and a monomial $m$.
**Output**: Test if $\mathsf{coeff}(p_C, m) = 0$ or not.

*Problem 4 (*$\mathsf{ExistExtMon}$*).* **Input**: An arithmetic circuit $C$ over $Z$, and a monomial $m$.
**Output**: Test if there is a monomial $M$ with non-zero coefficient in $p_C$ such that $M$ extends $m$; that is, $m|M$.

**Known Facts.** We list here some known results that we use in our constructions.

**Proposition 1 ( [6, 7]).** *Evaluating an arithmetic formula where the leaves are labelled* $\{-1, 0, 1\}$ *is in* $\mathsf{DLOG}$ *(even GapNC*$^1$*).*

**Proposition 2 ( [15]).** *Given $k$ ROFs in $n$ variables, there is a deterministic (non black-box) algorithm that checks whether they sum to zero or not. The running time of the algorithm is $n^{O(k)}$.*

**Proposition 3 (folklore).** *Given a formula $F$, a gate $g \in F$, and a variable $x$, checking whether $x \in \mathsf{var}_g$ is in* $\mathsf{DLOG}$*.*

**Proposition 4 (folklore).** *Given a rooted tree $T$, and two nodes $u, v$, the lowest common ancestor (LCA) of $u$ and $v$ can be found in* $\mathsf{DLOG}$*.*

## 3 Counting Monomials

We consider the $\mathsf{MonCount}$ problem for ROFs and OBPs. In both ROFs and OBPs, a monomial, once generated in a sub-formula/program, can be cancelled only by multiplication with a zero polynomial. We exploit this fact to obtain efficient algorithms for counting monomials in ROFs and OBPs.

**Theorem 1.** *Given a read-once formula $F$, $\mathsf{MonCount}(p_F)$ can be computed by an $AC^0$ circuit with oracle access to for $\mathsf{GapNC}^1$, and hence in $\mathsf{DLOG}$.*

*Proof.* We start with some notations. For gate $g$ in $F$, let $\#g$ denote the number of monomials in the polynomial $p_g$ computed at $g$. (The constant term, even if non-zero, does not count as a monomial.) Define the predicate $\mathsf{NZ}(g) = [p_g(0) \neq 0]$. The lemma below is proved in the appendix.

**Lemma 1.** *The language $L$ defined below is in $\mathsf{C_=NC}^1$:*

$$L = \{\langle F, g\rangle \mid F \text{ is an arithmetic formula, } g \text{ is a gate in } F, \text{ and } \mathsf{NZ}(g) = 0\}$$

Since $F$ is a read-once formula, we can compute the value of $\#f$ for each gate $f$ inductively, based on the structure of $F$ beneath $f$. When $f$ is a leaf node, it is labelled 0 or $\pm 1$ or $x_i$ for some $i$. Then $\#(0) = \#(\pm 1) = 0; \qquad \#(x_i) = 1.$

Now assume $f$ is not a leaf. Suppose $f = g + h$, then $g$ and $h$ are variable-disjoint read-once formulas. Since the monomials of $g$ and $h$ are distinct,

$$\#f = \#g + \#h; \tag{1}$$

Finally, suppose $f = g \times h$, then again $g$ and $h$ are variable-disjoint. Each pair of monomials $m$ in $p_f$ and $m'$ in $p_g$ gives rise to a monomial $mm'$ in $p_f$. In addition, if $p_g(0) \neq 0$, then each $m'$ also appears as a monomial in $p_f$; similarly for $p_h(0)$ and $m$. Thus

$$\#f = [\#g \times \#h] + [\#g \times \mathsf{NZ}(h)] + [\mathsf{NZ}(g) \times \#h]. \tag{2}$$

Using Equation 1 and Equation 2, we can transform the given read-once formula $F$ to a new formula $F'$ over $\mathbb{Z}$ that computes $\mathsf{MonCount}(F)$. The transformation is local, and can be done in $\mathsf{AC}^0$ with oracle access to $\mathsf{C_=NC}^1$. For each gate $f$ in $F$ the local transformation can be described as follows: If $f$ is a leaf gate, then relabel $f$ by $\#f$. If $f = g + h$, then apply Equation 1. If $f = g \times h$, using Equation 2 involves using $\#g$ and $\#h$ more than once, and so we do not get a formula. However, we can modify Equation 2 so that $\#f$ gets the structure of a formula, with oracle access to $\mathsf{NZ}$. We use the identity

$$\#(g \times h) = (\#g + \mathsf{NZ}(g)) \times (\#h + \mathsf{NZ}(h)) - (\mathsf{NZ}(g) \times \mathsf{NZ}(h)).$$

The values $\mathsf{NZ}(g)$ and $\mathsf{NZ}(h)$ can be obtained with oracle access to the language $L$ defined in Lemma 1. Now $\#g$ and $\#h$ are used only once.

Thus, from $F$ we construct a formula $F''$ where the leaves of $F''$ are labeled by constants $0, \pm 1$ or by the outputs of $\mathsf{C_=NC}^1$ oracle gates. Equivalently, in $\mathsf{AC}^0(\mathsf{C_=NC}^1)$, we can transform $F$ to formula $F'$ whose leaves are labeled by $0, \pm 1$. By construction, $F'$ is variable-free, and $\#p_F = \mathsf{val}(F')$. By Proposition 1, $\mathsf{val}(F')$ can be computed in $\mathsf{GapNC}^1$, completing the proof. □

*Remark 1.* The $\mathrm{AC}^0$ circuit constructed above needs oracle access mainly to $\mathsf{C_{=}NC^1}$ gates, which check whether a $\mathsf{GapNC^1}$ function is zero or not. Only the topmost oracle query requires the entire value of the $\mathsf{GapNC^1}$ function.

For any polynomial $p$, $p \equiv 0$ if and only if the constant term of $p$ is 0 and $\mathsf{MonCount}(\mathrm{p})$ is 0. Hence, from Theorem 1 and Lemma 1, we have the following:

**Corollary 1.** *In the non-blackbox setting,* ACIT *on ROFs is in the* GapNC *hierarchy and hence in* DLOG.

We now show how to count monomials in OBPs. The approach used in Theorem 1 does not directly generalize to OBPs, *i.e.,* knowing MonCount at immediately preceding nodes is not enough to compute MonCount at a given node in an OBP. However, since every variable occurs at most once in an OBP, every path generating a monomial should pass through one of these edges. This allows us to keep track of the monomials at any given node of the OBP, given the monomial count of all of its predecessors.

We begin with some notations. Let $B$ be an occur-once BP on the set of variables $X$, and $u, v$ be any nodes in $B$. Let $c(u, v)$ be the constant term in $p(u, v)$. We define the predicate $\mathsf{NZ}(u, v) = [c(u, v) \neq 0]$.

We cannot directly use the strategy we used for ROFs, since even in an OBP, there can be cancellations due to the constant terms. We therefore identify edges critical for a polynomial. We say that edge $e = (w, u)$ of $B$ *is critical to* $v$ if (1) $\mathsf{label}((w, u)) \in X$; and (2) $B$ has a directed path $\rho$ from $u$ to $v$ consisting only of edges labeled by $\{-1, 1\}$.

We have the following structural property for the monomials in $p(s, v)$:

**Lemma 2.** *In an occur-once OBP $B$ with start node $s$, for any node $v$ in $B$,*

$$p(s, v) = c(s, v) + \sum_{(w,u)\ \textit{critical to } v} p(s, w) \cdot \mathsf{label(w, u)} \cdot \mathsf{c(u, v)} \ .$$

For nodes $w, u, v$ where $(w, u)$ is an edge, define the predicate $\mathsf{critical}(\langle \mathsf{w, u} \rangle, \mathsf{v}) = [(w, u) \text{ is critical for } v]$. Using this and Lemma 2, we can show:

**Lemma 3.** *In an occur-once OBP $B$ with start node $s$, for any node $v$ in $B$,*

$$\#p(s, v) = \sum_{e=(w,u)} \mathsf{critical}(\langle \mathsf{w, u} \rangle, \mathsf{v}) \cdot \big(\#\mathsf{p(s, w)} + \mathsf{NZ(s, w)}\big) \cdot \mathsf{NZ(u, v)}.$$

*Proof.* Consider the expression $p(s, w) \times \mathsf{label(w, u)}$, where $(w, u)$ is an edge critical to $v$. Then $\mathsf{label(w, u)}$ is in $X$, and multiplies every monomial in $p(s, w)$. Hence every monomial of $p(s, w)$ contributes a monomial to $p(s, w) \times \mathsf{label(w, u)}$. Further, if $c(s, w) \neq 0$, then $c(s, w) \times \mathsf{label(w, u)}$ too contributes a monomial. Hence

$$\#[p(s, w) \times \mathsf{label(w, u)}] = \#\mathsf{p(s, w)} + \mathsf{NZ(s, w)} \ .$$

Using this observation along with Lemma 2 completes the proof. □

If $w$ is not in a layer to the left of $v$, then $(w, u)$ cannot be critical to $v$, and so $\#p(s, w)$ is not required while computing $\#p(s, v)$. Hence we can sequentially evaluate $\#p(s, v)$ for all nodes $v$ in layers going left to right, provided we have all the values $\mathsf{NZ}(s, w)$ and $\mathsf{critical}(\langle \mathsf{w}, \mathsf{u} \rangle, \mathsf{v})$.

**Lemma 4.** *The languages $L_1, L_2$ defined below are both in $\mathsf{C_=L}$.*

$$L_1 = \{\langle B, u, v \rangle \mid B \text{ is an OBP, } u, v \text{ are nodes in } B, \text{ and } \mathsf{NZ}(u, v) = 0.\}$$

$$L_2 = \left\{ \langle B, u, v, w \rangle \mid \begin{array}{l} B \text{ is an OBP, } u, v, w \text{ are nodes in } B, \text{ and} \\ \mathsf{critical}(\langle \mathsf{w}, \mathsf{u} \rangle, \mathsf{v}) = 1. \end{array} \right\}$$

From Lemma 3, the comment following it, and Lemma 4, we obtain a polynomial time algorithm to count the monomials in $p_B$.

**Theorem 2.** *Given an occur-once branching program $B$, the number of monomials in $p_B$ can be computed in $\mathsf{P}$.*

With a little bit of care, we can obtain the following stronger result:

**Theorem 3.** *Given an occur-once branching program $B$, the number of monomials in $p_B$ can be computed in the GapL hierarchy and hence in $\mathsf{NC}^2$.*

As in Corollary 1, using Theorem 3 and Lemma 4, we have:

**Corollary 2.** *In the non-blackbox setting, $\mathsf{ACIT}$ on OBPs is in the $\mathsf{GapL}$ hierarchy and hence in $\mathsf{NC}^2$.*

## 4 Zero-test on a Monomial Coefficient (ZMC)

From [10], $\mathsf{ZMC}$ is known to be in the second level of $\mathsf{CH}$ and hard for the class $\mathsf{C_=P}$. For the case of multilinear BPs MBPs, we show that $\mathsf{ZMC}$ exactly characterizes the complexity class $\mathsf{C_=L}$.

**Theorem 4.** *Given a BP $B$ computing a multilinear polynomial $p_B$, and given a multilinear monomial $m$, the coefficient of $m$ in $p_B$ can be computed in $\mathsf{GapL}$. Hence $\mathsf{ZMC}$ for multilinear BPs is complete for $\mathsf{C_=L}$.*

*Proof.* We first show that $\mathsf{ZMC}$, even for OBPs, is hard for $\mathsf{C_=L}$. A complete problem for $\mathsf{C_=L}$ is: does a BP $B$ with labels from $\{-1, 0, 1\}$ evaluate to 0? Add a node $t'$ as the new target node, and add edge $t \to t'$ labeled $x$ to get $B'$. Then $B'$ is an OBP, and $(B', x) \in \mathsf{ZMC}$ if and only if $B$ evaluates to 0.

Now we show the upper bound. We show that given an MBP $B$ computing a multilinear polynomial $p_B$, and given a multilinear monomial $m$, the coefficient of $m$ in $p_B$ can be computed in GapL. This implies that $\mathsf{ZMC}$ is in $\mathsf{C_=L}$.

Let $S \subseteq [n]$ be such that $m = m_S$. Let $p_B = \sum_{T \subseteq [n]} \mathsf{coeff}(p_B, m_T) m_T$. We are interested in $\mathsf{coeff}(p_B, m_S)$. The idea is to construct a branching program $B'$ computing a univariate polynomial, and a monomial $m'$, such that $m \in p_B$ if and only if $m' \in p_{B'}$. We obtain $B'$ by relabelling the edges of $B$ as follows:

| label in $B$ | constant $c$ | $x_i$ for $i \in S$ | $x_i$ for $i \notin S$ |
|---|---|---|---|
| label in $B'$ | constant $c$ | $y$ | 0 |

$B'$ now computes a univariate polynomial $p_{B'}$ in $y$. The coefficient $c_S$ of $m$ in $p_B$ is equal to the coefficient of $y^{|S|}$ in $p_{B'}$. To see this, note that

$$p_B = \sum_{T \subseteq [n]} \mathsf{coeff}(p_B, m_T) m_T = \sum_{T \subseteq S} \mathsf{coeff}(p_B, m_T) m_T + \sum_{T \not\subseteq S} \mathsf{coeff}(p_B, m_T) m_T$$

The substitution described above sends the second sum to zero in $B'$. Hence,

$$p_{B'}(y) = \sum_{T \subseteq S} \mathsf{coeff}(p_B, m_T) y^{|T|} = \sum_{j=0}^{|S|} \left( \sum_{\substack{T \subseteq S \\ |T|=j}} \mathsf{coeff}(p_B, m_T) \right) y^j$$

The only monomial in $p_B$ that generates $y^{|S|}$ in $p_{B'}$ is $\prod_{i \in S} x_i = m_S$.

(This argument only requires that $p_B$ be multilinear; we do not need $B$ to be occur-once or even read-once.)

Thus the problem now reduces to computing the coefficient of $y^{|S|}$ in $B'$, which is a branching program over just one input variable. A standard construction allows us to explicitly construct all coefficients of $p_{B'}(y)$ in another branching program $B''$. For completeness, we describe the construction of $B''$. For each node $v$ in $B'$, $B''$ has $|S|+1$ nodes $v_0, \ldots, v_{|S|}$, with the intention that $v_i$ should compute the coefficient of $y^i$ in the polynomial $p_{B'}(s, v)$. The start node of $B''$ is the node $s_0$, and the final node is $t_{|S|}$. If edge $(u, v)$ has label $y$ in $B'$, we include the edges $(u_i, v_{i+1})$ with label 1, for $0 \leq i < |S|$, in $B''$. If edge $(u, v)$ has label $\ell \neq y$ in $B'$, we include the edges $(u_i, v_i)$ with label $\ell$, for $0 \leq i \leq |S|$ , in $B''$. By induction on the structure of $B'$, we see that the value computed by $B''$ at $t_{|S|}$ is the coefficient of $y^{|S|}$ in $p_{B'}(s, v)$.

The above transformation from $B'$ to $B''$ can be done in DLOG. Since $B''$ is variable-free, it can be evaluated in GapL. Composing these procedures, we obtain a GapL procedure for computing the coefficient of $m$ in $p_B$. □

The upper bound above also applies to ROFs, since ROFs can be converted to OBPs by a standard construction. However, with a careful top-down algorithm, we get a stronger upper bound of DLOG for ZMC on ROFs. (See Appendix.)

**Theorem 5.** *Given a read-once formula $F$ computing a polynomial $p_F$, and given a multilinear monomial $m$, the coefficient of $m$ in $p_F$ can be computed in* DLOG. *Hence* ZMC *for ROFs is in* DLOG.

For ROFs, the lower bound proof in Theorem 4 can be modified to show that ZMC on ROFs is hard for $\mathsf{C}_=\mathsf{NC}^1$. It is natural to ask whether there is a matching upper bound. In our construction above, we need to compute predicates of the form $[x \in \mathsf{var}_\mathsf{g}]$. If these can be computed in $\mathsf{NC}^1$ for ROFs, then the monomial coefficients can be computed in $\mathsf{GapNC}^1$ and hence the upper bound of ZMC can be improved to $\mathsf{C}_=\mathsf{NC}^1$. However, this depends on the specific encoding of the input formula. In the standard pointer representation, the problem models reachability in out-degree-1 directed acyclic graphs, and is as hard as DLOG.

## 5 Checking existence of monomial extensions

We now address the problem ExistExtMon. Given a monomial $m$, one wants to check if the polynomial computed by the input arithmetic circuit has a monomial $M$ that extends $m$ (that is, with $m|M$). This problem is seemingly harder than ZMC, and hence the bound of Theorem 4 does not directly apply to ExistExtMon. We show that ExistExtMon for OBPs is in the GapL hierarchy.

**Theorem 6.** *The following problem lies in the* GapL *hierarchy: Given an occur-once branching program $B$ and a multilinear monomial $m$, check whether $p_B$ contains any monomial $M$ such that $m|M$.*

*Proof.* (Sketch) Let $S \subseteq [n]$ be such that $m = m_S$. If $S = \emptyset$, then this amounts to checking if $p_B \not\equiv 0$. By Corollary 2, this is in the GapL hierarchy. So now assume that $S \neq \emptyset$. We call an edge labelled by a variable from $S$ a "bridge". The algorithm is as follows:

**if** $\exists i \in S$ such that $x_i$ does not appear in $B$ at all **then**
  Output NO and halt.
**else if** $\exists$ layer $l$ with more than one bridge to layer $l+1$ **then**
  Output NO and halt.
**else**
  For each layer $l$ that has a bridge $e$ to layer $l+1$ in $B$, remove all edges except $e$. Call the branching program thus obtained $B'$.
**end if**
Output $\overline{\mathsf{ACIT}(p_{B'})}$ and halt.

The correctness is straightforward; see Appendix for details. By Corollary 2, the above algorithm is in the GapL hierarchy. □

The above bound can be brought down to DLOG for the case of ROFs.

**Theorem 7.** *The following problem is in* DLOG*: Given a read-once formula $F$ computing a polynomial $p_F$, and given a multilinear monomial $m$, check whether $p_F$ contains any monomial $M$ such that $m|M$.*

*Proof.* (Sketch) Let $S \subseteq [n]$ be such that $m = m_S$. If $S = \emptyset$, then we need to check if $p_F \not\equiv 0$; by Corollary 1, this is in DLOG. So now assume that $S \neq \emptyset$. As for BPs, we transform $F$ to a new formula $F'$, by cutting off sub-formulas addititvely related to variables in $m$, and then check $\mathsf{ACIT}(F')$. The transformation from $F$ to $F'$ can be done in DLOG (using Proposition 4). Then by Corollary 1, the algorithm can be implemented in DLOG. See the Appendix for details. □

## 6 Read-twice Formulas: multilinearity and identity tests

In this section we consider read-twice formulas, and the problems MLIN and ACIT on such formulas. The individual degree of a variable in a polynomial $p$ computed by read-twice formula $F$ is bounded by two. Thus, multilinearity testing boils down to testing if the second order partial derivative of $x_i$ is zero

for every variable $x_i$. Note that the partial derivative of $x_i$ is a polynomial in $n-1$ variables; thus MLIN reduces to $n$ instances of ACIT on $n-1$ variables. Our approach is to use the inductive structure of a read-twice polynomial to test these partial derivatives for zero, using the knowledge of multilinearity of gates at the lower levels. As an aid in this computation, we also check, for each gate $g$ and each variable $x$, whether $x$ survives in $p_g$.

**Theorem 8.** *For read-twice formulas, the problems* ACIT, MLIN, *and* $\mathsf{ExistExtMon}(\phi, x)$ *(where $\phi$ is the input formula and $x$ is a single variable in it) are in P.*

*Proof.* Let $\phi$ be the given read-twice formula. Without loss of generality, assume that $\phi$ is strictly alternating. That is, inputs to a + gate are either leaves or are $\times$ gates, and inputs to a $\times$ gate are either leaves or are + gates.

We proceed by induction on the structure of $\phi$. We iteratively compute, for each gate $g$ in $\phi$ and each variable $x \in X$, the following predicates: $\mathsf{ACIT}(g) = [p_g \equiv 0]$; $\mathsf{ExistExtMon}(g, x) = [p_g$ has a monomial containing $x]$; and $\mathsf{MLIN}(g) = [p_g$ is multilinear$]$. (We say that $x$ survives in $g$ if $\mathsf{ExistExtMon}(g, x) = 1$.) The base case is when $\phi$ consists of a single gate $g$ that is labelled $L \in X \cup \{0, +1, -1\}$. Then $\mathsf{ACIT}(g) = 1$ if and only if $L = 0$, $\mathsf{MLIN}(g) = 1$ always, and $\mathsf{ExistExtMon}(g, x) = 1$ if and only if $L = x$.

Now assume that for every gate $u$ below the root gate of $\phi$, the above predicates have been computed and stored. Let $f$ be the root gate of $\phi$. We show how to compute these predicates at $f$. The order in which we compute them depends on whether $f$ is $\times$ or a + gate.

First, consider $f = g \times h$. We compute the predicates in the order below.

1. $\mathsf{ACIT}(f) = \mathsf{ACIT}(g) \vee \mathsf{ACIT}(h)$.
2. $\mathsf{MLIN}(f)$: If $f$ is identically zero, then it is vacuously multilinear. Otherwise, for it to be multilinear, it must be the product of two (non-zero) multilinear polynomials in disjoint sets of variables. Thus

$$\mathsf{MLIN}(f) = \mathsf{ACIT}(f) \vee \left[ \mathsf{MLIN}(g) \wedge \mathsf{MLIN}(h) \wedge \left( \bigwedge_{x \in X} [\neg \mathsf{ExistExtMon}(g, x) \vee \neg \mathsf{ExistExtMon}(h, x)] \right) \right]$$

   Note that the $\mathsf{ACIT}(f)$ term is necessary, since $f$ can be multilinear even if, for instance, $g$ is not, provided $h \equiv 0$.
3. $\mathsf{ExistExtMon}(f, x) = \neg \mathsf{ACIT}(f) \wedge [\mathsf{ExistExtMon}(g, x) \vee \mathsf{ExistExtMon}(h, x)]$. ($x$ appears in $p_f$ if and only if $p_f \not\equiv 0$ and $x$ appears in at least one of $p_g, p_h$.)

Next, consider $f = g + h$. We compute the predicates in the order below.

1. $\mathsf{MLIN}(f) = \mathsf{MLIN}(g) \wedge \mathsf{MLIN}(h)$.
   (Since $f$ is read-twice, a non-multilinear monomial in $g$ cannot get cancelled by a non-multilinear monomial in $h$. Thus, $f$ is multilinear only if both $g$ and $h$ are. The converse is trivially true.)

2. $\mathsf{ExistExtMon}(f, x)$: See below.
3. $\mathsf{ACIT}(f) = \bigwedge_{x \in X} \neg \mathsf{ExistExtMon}(f, x)$.

We now complete the description for computing $\mathsf{ExistExtMon}(f, x)$ when $f = g + h$. If $x$ survives in neither $g$ nor $h$, then it does not survive in $f$. But if it survives in exactly one of $g, h$, it cannot get cancelled, so it survives in $f$. Thus

$$\mathsf{ExistExtMon}(g, x) \vee \mathsf{ExistExtMon}(h, x) = 0 \Longrightarrow \mathsf{ExistExtMon}(f, x) = 0$$
$$\mathsf{ExistExtMon}(g, x) \oplus \mathsf{ExistExtMon}(h, x) = 1 \Longrightarrow \mathsf{ExistExtMon}(f, x) = 1$$

So now assume that $x$ survives in both $g$ and $h$. We can write the polynomials computed at $g, h$ as $p_g = \alpha x + \alpha'$ and $p_h = \beta x + \beta'$, where $\alpha', \beta'$ do not involve $x$; and we know that $\alpha \not\equiv 0$, $\beta \not\equiv 0$. We want to determine whether $\alpha + \beta \equiv 0$.

Since $x$ appears in $V_g$ and $V_h$, and since $f$ is read-twice, we conclude that $x$ is read exactly once each in $g$ and in $h$. Hence $\alpha, \beta$ also do not involve $x$.

We construct a formula computing $\alpha$ as follows: In the sub-formula rooted at $g$, let $\rho$ be the unique path from $x$ to $g$. For each $+$ gate $u$ on the path $\rho$, let $u'$ be the child of $u$ not on $\rho$; replace $u'$ by the constant 0. Thus we retain only the parts that multiply $x$; that is, we compute $\alpha x$. Setting $x = 1$ gives us a formula $G$ computing $\alpha$. A similar construction with the formula rooted at $h$ gives a formula $H$ computing $\beta$. Set $F = G + H$. Note that $F$ is also a read-twice formula, and it computes $\alpha + \beta$. Thus in this case $\mathsf{ExistExtMon}(f, x) = 1 \Leftrightarrow \mathsf{ACIT}(F) = 0$, so we need to determine $\mathsf{ACIT}(F)$.

Let $Y$ denote the set of variables appearing in $F$; $Y \subseteq X \setminus \{x\}$. Partition $Y$ into $A$ (variables occurring only in $G$), $B$ (variables occurring only in $H$), and $C$ (variables occurring in $G$ and $H$).

If $A \cup B = \emptyset$, then $Y = C$, and each variable in $F$ appears once in $G$ and once in $H$. That is, both $G$ and $H$ are read-once formulas. We can now determine $\mathsf{ACIT}(F)$ in time polynomial in the size of $F$ using Proposition 2.

If $A \cup B \neq \emptyset$, then let $y \in A$. If $y$ survives in $G$, it cannot get cancelled by anything in $H$, so it survives in $F$ and $F \not\equiv 0$. Similarly, if any $y \in B$ survives in $H$, then $F \not\equiv 0$. We briefly defer how to determine this and complete the high-level argument. If no $y \in A$ survives in $G$, and no $y \in B$ survives in $H$, then let $F' = G' + H'$ be the formula obtained from $F, G, H$ by setting variables in $A \cup B$ to 0. Clearly, the polynomial computed remains the same; thus $\alpha + \beta = p_F = p_F|_{A \cup B \leftarrow 0} = p_{F'}$. But $F'$ satisfies the previous case (with respect to $F'$, $A' \cup B' = \emptyset$), and so we can use Proposition 2 as before to determine $\mathsf{ACIT}(F') = \mathsf{ACIT}(F)$.

What remains is to describe how we determine whether a variable $y \in A$ survives in $G$. (The situation for $y \in B$ surviving in $H$ is identical.) We exploit the special structure of $G$: there is a path $\rho$ where all the $+$ gates have one argument 0 and the path ends in a leaf labeled 1. Let $\mathcal{T} = \{T_1, \ldots, T_\ell\}$ be the subtrees hanging off the $\times$ gates on $\rho$; let $u_i$ be the root of $T_i$. Note that each $T_i \in \mathcal{T}$ is a sub-formula of our input formula $\phi$, and hence by the iterative construction we know the values of the predicates $\mathsf{ACIT}$, $\mathsf{MLIN}$, $\mathsf{ExistExtMon}$ at gates in these sub-trees. In fact, we already know that $\mathsf{ACIT}(u_i) = 0$ for all $i$, since we are in the

situation where $\alpha \not\equiv 0$, and $\alpha = \prod_{i=1}^{\ell} p_{u_i}$. Hence, if $y$ appears in just one sub-tree $T_i$, then $\mathsf{ExistExtMon}(G, y) = \mathsf{ExistExtMon}(u_i, y)$. If $y$ appears in two sub-trees $T_i, T_j$, then $\mathsf{ExistExtMon}(G, y) = \mathsf{ExistExtMon}(u_i, y) \vee \mathsf{ExistExtMon}(u_j, y)$. □

## 7 Conclusion

Our results show that as expected, the complexity of $\mathsf{MonCount}, \mathsf{ZMC}$, and $\mathsf{ExistExtMon}$ reduce drastically for the case of severely restricted circuits. Ideally, we would like these problems to characterise complexity classes within P; we have partially succeeded in this. We leave open the question of extending these bounds for formulas and branching programs that are constant-read. It appears that this will require non-trivial modifications of our techniques.

## References


1. M. Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *FOCS*, pages 67–75, 2008.
2. E. Allender. Arithmetic circuits and counting complexity classes. In J. Krajicek, editor, *Complexity of Computations and Proofs*, Quaderni di Matematica Vol. 13, pages 33–72. Seconda Universita di Napoli, 2004. An earlier version appeared in the Complexity Theory Column, SIGACT News 28, 4 (Dec. 1997) pp. 2-15.
3. E. Allender, R. Beals, and M. Ogihara. The complexity of matrix rank and feasible systems of linear equations. *Computational Complexity*, 8(2):99–126, 1999.
4. M. Anderson, D. van Melkebeek, and I. Volkovich. Derandomizing polynomial identity testing for multilinear constant-read formulae. In *IEEE Conference on Computational Complexity*, pages 273–282, 2011. ECCC TR 17, 2010.
5. S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.
6. S. Buss. The Boolean formula value problem is in ALOGTIME. In *STOC*, pages 123–131, 1987.
7. S. Buss, S. Cook, A. Gupta, and V. Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM Journal of Computation*, 21(4):755–780, 1992.
8. H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic $\mathrm{NC}^1$ computation. *Journal of Computer and System Sciences*, 57:200–212, 1998.
9. S. Datta, M. Mahajan, B. V. R. Rao, M. Thomas, and H. Vollmer. Counting classes and the fine structure between $\mathrm{NC}^1$ and L. *Theoretical Computer Science*, 417:36–49, 2012.
10. H. Fournier, G. Malod, and S. Mengel. Monomials in arithmetic circuits: Complete problems in the counting hierarchy. In *STACS*, 2012. To appear.
11. M. J. Jansen, Y. Qiao, and J. M. N. Sarma. Deterministic black-box identity testing $\pi$-ordered algebraic branching programs. In *FSTTCS*, pages 296–307, 2010.
12. V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004.
13. P. Koiran and S. Perifel. The complexity of two problems on arithmetic circuits. *Theoretical Computer Science*, 389(1-2):172–181, 2007.
14. A. Shpilka and I. Volkovich. Read-once polynomial identity testing. In *STOC*, pages 507–516, 2008.
15. A. Shpilka and I. Volkovich. Improved polynomial identity testing for read-once formulas. In *APPROX-RANDOM*, pages 700–713, 2009.

## Appendix

We include here the proofs that were omitted in the main text due to space constraints.

*Proof.* (Of Lemma 1) Convert $F$ to formula $F'$ where all variables are set to 0, and $g$ is the output gate. Then $F'$ evaluates to $p_g(0)$, so we need to check if $F'$ evaluates to 0. By Proposition 1, this check can be performed in $\mathsf{C_{=}NC^1}$. □

*Proof.* (Of Lemma 2) Note that if edges $(w,u) \neq (w',u')$ are both critical to $v$, then the monomials in $p(s,w) \cdot \mathsf{label}(\mathsf{w},\mathsf{u})$ and $p(s,w') \cdot \mathsf{label}(\mathsf{w'},\mathsf{u'})$ will be disjoint, because $P$ is occur-once. (The variables labeling $(w,u)$ and $(w',u')$ make the monomials distinct.) Moreover, for any monomial $m$ in $p(s,v)$, there is exactly one critical edge $(w,u)$ such that the monomial $m$ has non-zero coefficient in the polynomial $p(s,w) \times \mathsf{label}(\mathsf{w},\mathsf{u})$. The critical edge corresponds to the last variable of the monomial to be "collected" en route to $v$ from $s$. This completes the proof. □

*Proof.* (Of Lemma 4) Delete from $B$ all edges with labels from $X$ to get a variable-free BP $B'$. Then $p_{B'}(u,v) = c_B(u,v)$. Checking whether $p_{B'}(u,v) = 0$ is the canonical complete problem for $\mathsf{C_{=}L}$. Hence $L_1$ is in $\mathsf{C_{=}L}$. To check membership in $L_2$, we need to check that $\mathsf{label}(\mathsf{w},\mathsf{u}) \in \mathsf{X}$ and that $v$ is reachable from $u$ in $B'$. This can be done in NLOG, which is contained in $\mathsf{C_{=}L}$. □

*Proof.* (Of Theorem 3 ) Starting from $B$, we construct another BP $B'$ as follows: $B'$ has a node $v'$ for each node $v$ of $B$. For each triple $w,u,v$ where $(w,u)$ is an edge in $B$, we check via oracles for $L_1$ and $L_2$ whether $(w,u)$ is critical to $v$ and whether $\mathsf{NZ}(u,v) = 1$. If both checks pass, we add an edge from $w'$ to $v'$. We also check whether $\mathsf{NZ}(s,w) = 1$, and if so, we add an edge from $s'$ to $v'$. (We do this for every $w,u$, so we may end up with multiple parallel edges from $s'$ to $v'$. To avoid this, we can subdivide each such edge added.) $B'$ thus implements the right-hand-side expression in Lemma 3. It follows that $p_{B'}(s',v')$ equals $\#p_B(s,v)$. Note that $B'$ can be constructed in logspace with oracle access to $\mathsf{C_{=}L}$. Also, since $B'$ is variable-free, it can be evaluated in $\mathsf{GapL}$. Hence $\#p_B$ can be computed in the GapL hierarchy. □

*Proof.* (Of Theorem 5) Let $\alpha(g,T)$ denote the coefficient of monomial $m_T$ in $p_g$. (That is, $\alpha(g,T) = \mathsf{coeff}(p_g, m_T)$.) Let $r$ be the output gate. Let $S \subseteq [n]$ be such that $m = m_S$. The goal is to compute $\alpha(r,S)$. First, we observe some properties of $\alpha$:

1. For any gate $g$ and any $T \subseteq [n]$, if $T \not\subseteq \mathsf{var_g}$, then $\alpha(g,T) = 0$.
2. For a leaf $g$ labelled $x_i$, $\alpha(g,T) = 1$ if $T = \{i\}$, 0 otherwise.
3. For a leaf $g$ labelled by a constant c, $\alpha(g,T)$=$c$ if $T = \emptyset$, 0 otherwise.
4. For an addition gate that computes $g + h$, $\alpha(g+h,T) = \alpha(g,T) + \alpha(h,T)$. And since $F$ is an ROF, at least one of $\alpha(g,T), \alpha(h,T)$ is zero.

5. For a product gate that computes $g \times h$,

$$\alpha(g \times h, T) = \alpha(g, T \cap \mathsf{var_g}) \cdot \alpha(\mathsf{h}, \mathsf{T} \cap \mathsf{var_h}) \cdot [\mathsf{T} \subseteq \mathsf{var_g} \cup \mathsf{var_h}].$$

This is because $\alpha(g \times h, T) = \sum_{Z \subseteq T} \alpha(g, Z)\alpha(h, T \setminus Z)$. But if either $Z \not\subseteq \mathsf{var_g}$ or $T \setminus Z \not\subseteq \mathsf{var_h}$, then $\alpha(g, Z) = 0$ or $\alpha(h, T \setminus Z) = 0$. Further, $F$ is a read once formula, so $\mathsf{var_g} \cap \mathsf{var_h} = \emptyset$, and $\mathsf{var_g}$ and $\mathsf{var_h}$ partition $\mathsf{var_{(g \times h)}}$. Hence $T$ must also be similarly partitioned.

Now we construct a formula $F'$ whose evaluation gives us $\alpha(r, S)$. $F'$ will recursively compute $\alpha(g, S \cap \mathsf{var_g})$ for each gate $g$. If $g$ is a leaf, we just use properties (2,3) to compute $\alpha(g, S \cap \mathsf{var_g})$ We show how to compute $\alpha(f, S \cap \mathsf{var_f})$ for an internal gate $f$ with children $g$ and $h$ knowing the values for $\alpha(g, S \cap \mathsf{var_g})$ and $\alpha(h, S \cap \mathsf{var_h})$:

– Case $f = g + h$:

$$\begin{aligned}
\alpha(f, S \cap \mathsf{var_f}) &= \alpha(g, S \cap \mathsf{var_f}) + \alpha(\mathsf{h}, \mathsf{S} \cap \mathsf{var_f}) \qquad \text{from property (4)} \\
&= \alpha(g, S \cap \mathsf{var_g})[\mathsf{S} \cap \mathsf{var_g} = \mathsf{S} \cap \mathsf{var_f}] \\
&+ \alpha(h, S \cap \mathsf{var_h})[\mathsf{S} \cap \mathsf{var_h} = \mathsf{S} \cap \mathsf{var_f}] \qquad \text{from property (1)}
\end{aligned}$$

– Case $f = g \times h$:

$$\alpha(f, S \cap \mathsf{var_f}) = \alpha(g, S \cap \mathsf{var_g}) \cdot \alpha(\mathsf{h}, \mathsf{S} \cap \mathsf{var_h}) \qquad \text{from properties (1,5)}$$

This gives us the formula $F'$ that computes $\alpha(r, S)$ at the topmost gate. By Proposition 1, $F'$ can be evaluated in GapNC$^1$. Constructing $F'$ from $F$ requires a local transformation at $+$ gates and computation of the predicates $[S \cap \mathsf{var_f} = \mathsf{S} \cap \mathsf{var_g}]$. By Proposition 3, these predicates can be computed in DLOG. □

*Proof.* (Details, for Theorem 6) Let $S \subseteq [n]$ be such that $m = m_S$. If $S = \emptyset$, then this amounts to checking if $p_B \not\equiv 0$. By Corollary 2, this is in the $\mathsf{GapL}$ hierarchy. So now assume that $S \neq \emptyset$. We call an edge that is labelled by a variable from $S$ a "bridge".

The algorithm is as follows:

**if** $\exists i \in S$ such that $x_i$ does not appear in $B$ at all **then**
  Output NO and halt.
**else if** $\exists$ layer $l$ with more than one bridge to layer $l+1$ **then**
  Output NO and halt.
**else**
  For each layer $l$ that has a bridge $e$ to layer $l+1$ in $B$, remove all edges except $e$. Call the branching program thus obtained $B'$.
**end if**
Output $\overline{\mathsf{ACIT}(p_{B'})}$ and halt.

We now show that $m_S$ has an extended monomial in $p_B$ if and only if the above algorithm outputs YES. If any of the variables of $m_S$ do not appear at all in $B$, then clearly an extension to $m_S$ cannot exist. So the algorithm rejects correctly. If there is a layer with more than one bridge to the next layer, then any path can go through at most one of these bridges. Since $B$ is occur-once, every path would compute a monomial with at least one variable from $m_S$ missing. So the algorithm correctly rejects. We are only interested in monomial extensions of $m_S$. So paths that do not go through all the bridges can be ignored. Hence we can safely delete all non-bridge edges in layers which have a bridge to the next layer. Thus $p_{B'}$ is a polynomial where each monomial is an extension of $m_S$.

By Corollary 2, the above algorithm is in the GapL hierarchy. □

*Proof.* (Details for Theorem 7) Let $S \subseteq [n]$ be such that $m = m_S$. If $S = \emptyset$, then this amounts to checking if $p_F \not\equiv 0$, which, by Corollary 1, is in DLOG. So now assume that $S \neq \emptyset$. As for BPs, we transform $F$ to a new formula $F'$ as follows:

**if** $\exists x_i \in S$ such that $x_i$ does not appear in $F$ at all **then**
  Output NO and halt.
**else if** $\exists x_i, x_j \in S$, $i \neq j$, with LCA$(x_i, x_j)$ in $F$ labeled $+$ **then**
  Output NO and halt.
**else**
  For every $x_i \in S$, and every $+$ gate $g$ on the unique leaf-to-root path $\gamma$ from $x_i$, replace the input of $g$ not on the path $\gamma$ by 0.
  Let $F'$ be the resulting formula.
**end if**
Output $\overline{\mathsf{ACIT}(F')}$.

We show correctness of the above algorithm. Since $F$ is read-once, if any of the two variables in $S$ have a $+$ gate as their lowest common ancestor, then $m$ cannot appear as a monomial in $F$. The second step in the algorithm removes all sub-formulas that are additively related to some variable $x_i$ in $S$. This implies that every monomial produced by $F'$ has $m$ as a factor. Also, any monomial $m'$ of $p_F$ with $m|m'$ has the same coefficient in $p_{F'}$ as in $p_F$. Thus, the resulting formula $F'$ computes a polynomial that contains exactly all monomials $m'$ of $p_F$ such that $m|m'$. This proves the correctness.

For the complexity bound, we note that the transformation from $F$ to $F'$ can be done in DLOG (using Proposition 4). Then by Corollary 1, the overall algorithm can be implemented in DLOG. □