comput. complex. **24** (2015), 1 – 30
© Springer Basel 2014
1016-3328/15/010001-30
published online January 18, 2014
DOI 10.1007/s00037-013-0079-3

computational complexity

MONOMIALS IN ARITHMETIC CIRCUITS: COMPLETE PROBLEMS IN THE COUNTING HIERARCHY

Hervé Fournier, Guillaume Malod, and Stefan Mengel

Abstract. We consider the complexity of two questions on polynomials given by arithmetic circuits: testing whether a monomial is present and counting the number of monomials. We show that these problems are complete for subclasses of the counting hierarchy which had few or no known natural complete problems before. We also study these questions for circuits computing multilinear polynomials and for univariate multiplicatively disjoint circuits.

Keywords. Arithmetic circuits, counting problems, polynomials.

Subject classification. 68Q15, 68Q17, 03D15.

1. Introduction

We study the complexity of two problems on polynomials represented by arithmetic circuits. The first one is to decide whether a given monomial has zero coefficient, while the second consists in counting the number of monomials. We characterize their complexity using the counting hierarchy.

The counting hierarchy refers to the family of classes $\mathsf{PP} \cup \mathsf{PP}^{\mathsf{PP}} \cup \mathsf{PP}^{\mathsf{PP}^{\mathsf{PP}}} \cup \ldots$. It has appeared in several recent papers. For example, Bürgisser (2009) uses these classes to connect computing integers to computing polynomials, while Jansen & Santhanam (2011)—building on results by Koiran & Perifel (2011)—use them to derive lower bounds from derandomization. This hierarchy was originally introduced by Wagner (1986) to classify the complexity of combinatorial problems. Curiously, after Wagner's paper and another by Torán (1988), this original motivation of the counting hierarchy has to the best of our knowledge not been pursued for more than twenty years. Instead, research focused on structural properties and the connection to threshold circuits (Allender & Wagner 1993). As a result, there are very few natural complete problems for classes in the counting hierarchy: for instance, Kwisthout *et al.* (2011) give "the first problem with a practical application that is shown to be $\mathsf{FP}^{\mathsf{PP}^{\mathsf{PP}}}$-complete." The related class $\mathsf{C_{=}P}$ appears to have no natural complete problems at all (Hemaspaandra & Ogihara 2002, p. 293). It is, however, possible to define generic complete problems by starting with a #P-complete problem and considering the variant where an instance and a positive integer are provided, and the question is to decide whether the number of solutions for this instance is equal to the integer. We consider these problems to be counting problems disguised as decision problems and thus not as natural complete problems for $\mathsf{C_{=}P}$, in contrast to the questions studied here. Note that the corresponding logspace counting class $\mathsf{C_{=}L}$ is known to have interesting complete problems from linear algebra (Allender *et al.* 1999).

In this paper, we follow Wagner's original idea and show that the counting hierarchy is a helpful tool to classify the complexity of several natural problems on arithmetic circuits by showing complete problems for the classes $\mathsf{PP}^{\mathsf{PP}}$, $\mathsf{PP}^{\mathsf{NP}}$ and $\mathsf{C_{=}P}$. Mundhenk *et al.* (2000) present several complete problems for the class $\mathsf{NP}^{\mathsf{PP}}$, in particular from the AI/planning literature. The claim in Hemaspaandra & Ogihara (2002, p. 293) that Mundhenk *et al.* (2000) provide natural complete problems for $\mathsf{PP}^{\mathsf{NP}}$ is a typo corrected on the book's errata page (`http://www.cs.rochester.edu/u/lane/=companion/errata.pdf`).

The common setting of these problems is the use of circuits or straight-line programs to represent polynomials. Such a representation can be much more efficient than giving the list of monomials, but common operations on polynomials may become more difficult. An important example is the question of determining whether the given polynomial is identically zero. This is easy to do when the polynomial is given as a list of monomials. When the polynomial is given as a circuit, however, this problem, called ACIT for *arithmetic circuit identity testing*, is not known to be in P, though it is in coRP. In fact, derandomizing this problem would imply circuit lower bounds, as shown in Heintz & Schnorr (1980) and Kabanets & Impagliazzo (2004). This question thus plays a crucial part in complexity, and it is natural to consider other problems on polynomials represented as circuits. In this article, we consider mainly two questions.

The first main problem, called ZMC for *zero monomial coefficient*, is to decide whether a given monomial in a circuit has coefficient 0 or not. This problem has already been studied by Koiran & Perifel (2007). They showed that when the formal degree of the circuit is polynomially bounded, the problem is complete for $\mathsf{P}^{\#\mathsf{P}}$. Unfortunately, this result is not fully convincing, because it is formulated with the rather obscure notion of strong non-deterministic Turing reductions. We remedy this situation by proving a completeness result for the class $\mathsf{C}_=\mathsf{P}$ under more traditional logarithmic-space many-one reductions. This also provides a natural complete problem for this class. Koiran & Perifel (2007) also considered the general case of ZMC, where the formal degree of the circuit is not bounded. They showed that ZMC is in the counting hierarchy. We provide a better upper bound by proving that ZMC is in $\mathsf{coRP}^{\mathsf{PP}}$. We finally study the case of monotone circuits and show that the problem is then coNP-complete.

The second main problem is to count the number of monomials in the polynomial computed by a circuit. In the general case, this natural counting problem turns out to be $\mathsf{PP}^{\mathsf{PP}}$-complete, and the hardness holds even for weak circuits such as depth-4 formulas. We thus obtain another natural complete problem, in this case for the second level of the counting hierarchy. We remark that if a polynomial bound is given on the number of monomials, both the problem ZMC and the one of counting monomials become easy since an explicit description of the polynomial can be computed in polynomial time (Garg & Schost 2009; Klivans & Spielman 2001). The related problem of enumerating the monomials of a given polynomial, in the black-box model, is addressed in Strozecki (2013).

Then we study the two above problems in the case of circuits computing multilinear polynomials. We show that our first problem becomes equivalent to the fundamental problem ACIT and that counting monomials becomes PP-complete.

Finally, we consider the case of univariate multiplicatively disjoint circuits. We show that these problems and several related ones are equivalent and complete for LOGCFL in the monotone case, or close to $\mathsf{C_{=}LOGCFL}$ in the general case.

2. Preliminaries

Arithmetic circuits. An *arithmetic circuit* is a labeled directed acyclic graph (DAG) consisting of vertices or gates with in degree or fanin 0 or 2. The gates with fanin 0 are called input gates and are labeled with -1 or variables $X_1, X_2, \ldots, X_n$. The gates with fanin 2 are called computation gates and are labeled with $\times$ or $+$. We can also consider circuits where computation gates may receive more than two edges, in which case we say that they have *unbounded fanin.* The polynomial computed by an arithmetic circuit is defined in the obvious way: an input gate computes the value of its label, a computation gate computes the product or the sum of its children's values, respectively. We assume that a circuit has only one sink which we call the output gate. We say that the polynomial computed by the circuit is the polynomial computed by the output gate. The *size* of an arithmetic circuit is the number of gates. The *depth* of a circuit is the length of the longest path from an input gate to the output gate in the circuit. A formula is an arithmetic circuit whose underlying graph is a tree. Finally, a circuit or formula is called *monotone* if, instead of the constant -1, only the constants 0 and 1 are allowed. When an arithmetic circuit is the input of a problem, we consider that it is given as a graph with labels on the vertices, for instance as an adjacency list.

It is common to consider so-called *degree-bounded* arithmetic circuits, for which the degree of the computed polynomial is bounded polynomially in the number of gates of the circuit. In our opinion, this kind of degree bound has two problems. One is that computing the degree of a polynomial represented by a circuit is suspected to be hard (see Section 6 and Allender *et al.* 2009; Kayal

& Saha 2011; Koiran & Perifel 2007), so problems defined with this degree bound must often be promise problems. The other problem is that the bound on the degree does not bound the size of computed constants, which by iterative squaring can have exponential bitsize. Thus, even evaluating circuits on a Turing machine becomes intractable. The paper by Allender *et al.* (2009) discusses problems that result from this. To avoid all these complications, instead of bounding the degree of the computed polynomial, we choose to bound the formal degree of the circuit or equivalently to consider multiplicatively disjoint circuits. A circuit is called *multiplicatively disjoint* if, for each ×-gate, its two input subcircuits are disjoint from one another. See Malod & Portier (2008) for a discussion of degree, formal degree and multiplicative disjointness and how they relate.

Throughout the paper, we assume that the underlying field has characteristic zero.

Complexity classes. We assume that the reader is familiar with basic concepts of computational complexity theory (see e.g., Arora & Barak 2009). All reductions in this paper will be logspace many-one unless stated otherwise.

We consider different counting decision classes in the counting hierarchy (Wagner 1986). These classes are defined analogously to the quantifier definition of the polynomial hierarchy but, in addition to the quantifiers ∃ and ∀, the quantifiers C, $\mathsf{C}_=$ and $\mathsf{C}_{\neq}$ are used.

Definition 2.1. *Let $\mathcal{C}$ be a complexity class containing* P.

- *$A \in \mathsf{C}\mathcal{C}$ if and only if there is $B \in \mathcal{C}$, $f \in \mathsf{FP}$ and a polynomial p such that*

$$x \in A \Leftrightarrow \left|\left\{y \in \{0,1\}^{p(|x|)} \mid (x,y) \in B\right\}\right| \geq f(x),$$

- *$A \in \mathsf{C}_=\mathcal{C}$ if and only if there is $B \in \mathcal{C}$, $f \in \mathsf{FP}$ and a polynomial p such that*

$$x \in A \Leftrightarrow \left|\left\{y \in \{0,1\}^{p(|x|)} \mid (x,y) \in B\right\}\right| = f(x),$$

- ○ *$A \in \mathsf{C}_{\neq}\mathcal{C}$ if and only if there is $B \in \mathcal{C}$, $f \in \mathsf{FP}$ and a polynomial p such that*

$$x \in A \Leftrightarrow \left|\left\{y \in \{0,1\}^{p(|x|)} \mid (x,y) \in B\right\}\right| \neq f(x).$$

Observe that $\mathsf{C}_{\neq}\mathcal{C} = \mathsf{coC}_{=}\mathcal{C}$ with the usual definition $\mathsf{co}\mathcal{C} = \{L^c \mid L \in \mathcal{C}\}$, where L^c is the complement of L. That is why the quantifier $\mathsf{C}_{\neq}$ is often also written as $\mathsf{coC}_{=}$, so $\mathsf{C}_{\neq}\mathsf{P}$ is sometimes called $\mathsf{coC}_{=}\mathsf{P}$.

The counting hierarchy CH consists of the languages from all classes that we can get from P by applying the quantifiers $\exists$, $\forall$, C, $\mathsf{C}_{=}$ and $\mathsf{C}_{\neq}$ a constant number of times. Observe that with the definition above $\mathsf{PP} = \mathsf{CP}$. Torán (1991) proved that this connection between PP and the counting hierarchy can be extended and that there is a characterization of CH by oracles similar to that of the polynomial hierarchy. We state some such characterizations which we will need later on, followed by other technical lemmas.

Lemma 2.2 (Torán 1991). $\mathsf{PP}^{\mathsf{NP}} = \mathsf{C}\exists\mathsf{P}$.

Lemma 2.3. $\mathsf{PP}^{\mathsf{PP}} = \mathsf{CC}_{\neq}\mathsf{P}$.

Proof (Lemma 2.3). This is not stated in Torán (1991) nor is it a direct consequence, because Torán does not consider the $\mathsf{C}_{\neq}$-operator. It can be shown with similar techniques and we give a proof for completeness. We show that $\mathsf{CC}_{\neq}\mathsf{P} = \mathsf{CCP}$, the claim then follows, because $\mathsf{CCP} = \mathsf{PP}^{\mathsf{PP}}$ by Torán (1991).

The direction from left to right is straightforward: From the definition we have $\mathsf{CC}_{\neq}\mathsf{P} \subseteq \mathsf{PP}^{\#\mathsf{P}}$. By binary search we have $\mathsf{PP}^{\#\mathsf{P}} = \mathsf{PP}^{\mathsf{PP}} = \mathsf{CCP}$. The other direction needs a little more work. Let $L \in \mathsf{CCP}$, there are $A \in \mathsf{P}$,$f, g \in \mathsf{FP}$ and a polynomial p such that

$$\begin{aligned} x \in L \quad &\Leftrightarrow \quad \text{there are more than } f(x) \text{ values } y \in \{0,1\}^{p(|x|)} \\ &\qquad \text{such that } \left|\left\{z \in \{0,1\}^{p(|x|)} \mid (x,y,z) \in A\right\}\right| \geq g(x,y) \\ &\Leftrightarrow \quad \text{there are more than } f(x) \text{ values } y \in \{0,1\}^{p(|x|)} \\ &\qquad \text{such that } \forall v \in \{1,\ldots,2^{p(|x|)}\}: \end{aligned}$$

$$\begin{aligned}
&(2.4) && \left|\left\{z \in \{0,1\}^{p(|x|)} \mid (x,y,z) \in A\right\}\right| \neq g(x,y) - v \\
&&\Leftrightarrow\ & \text{there are more than } 2^{p(|x|)}(2^{p(|x|)} - 1) + f(x) \text{ pairs} \\
&&& (y,v) \text{ with } y \in \{0,1\}^{p(|x|)} \text{ and } v \in \{1,\dots,2^{p(|x|)}]\} \\
&&& \text{such that:} \\
&(2.5) && \quad \left|\left\{z \in \{0,1\}^{p(|x|)} \mid (x,y,z) \in A\right\}\right| \neq g(x,y) - v.
\end{aligned}$$

From statement (2.5), we directly get $L \in \mathsf{CC}_{\neq}\mathsf{P}$ and thus the claim. To see the last equivalence, we define

$$r(x,y) := \left|\left\{z \in \{0,1\}^{p(|x|)} \mid (x,y,z) \in A\right\}\right|.$$

Fix x, y, then obviously $r(x,y) \neq g(x,y) - v$ for all but at most one v. It follows that of the pairs (y,v) in the last statement $2^{p(|x|)}(2^{p(|x|)} - 1)$ always lead to inequality. So statement (2.5) boils down to the question how many y there are such that there is no v with $r(x,y) = g(x,y) - v$. We want these to be at least $f(x)$, so we want at least $2^{p(|x|)}(2^{p(|x|)} - 1) + f(x)$ pairs such that $r(x,y) \neq g(x,y) - v$. □

Lemma 2.6 (Green 1993). $\exists\mathsf{C}_{\neq}\mathsf{P} = \mathsf{C}_{\neq}\mathsf{P}$.

Lemma 2.7 (Schönhage 1979). *For a large enough constant $c > 0$, it holds that for any integers n and x with $|x| \leqslant 2^{2^n}$ and $x \neq 0$, the number of primes p smaller than 2^{cn} such that $x \not\equiv 0 \mod p$ is at least $2^{cn}/cn$.*

Lemma 2.8 (Hemaspaandra & Ogihara 2002, p. 81). *The classes $\mathsf{PP}^{\mathsf{BPP}^X}$ and PP^X are equal for every oracle X.*

3. Zero monomial coefficient

We first consider the question of deciding if a single specified monomial occurs in a polynomial. In this problem and others regarding monomials, a monomial is encoded by giving the variable powers in binary. All input integers are given in binary unless stated otherwise.

ZMC
Input: Arithmetic circuit C, monomial m.
Problem: Decide if m has the coefficient 0 in the polynomial computed by C.

Recall that we always assume the underlying field has characteristic zero. The problem ZMC in positive characteristic was considered by Koiran & Perifel (2007).

Theorem 3.1. ZMC *is* $\mathsf{C_=P}$*-complete for both multiplicatively disjoint circuits and formulas.*

Proof. Using standard reduction techniques from the proof of #P-completeness for the permanent (see for example Arora & Barak 2009), one can define the following generic $\mathsf{C_=P}$-complete problem, as mentioned in the introduction.

PER$_=$
Input: Matrix $A \in \{0, 1, -1\}^{n\times n}$, $d \in \mathbb{N}$.
Problem: Decide if $\textsc{per}(A) = d$.

Therefore, for the hardness of ZMC it is sufficient to show a reduction from PER$_=$.

On input $A = (a_{ij})$ and d we compute the formula $Q := \prod_{i=1}^{n}\left(\sum_{j=1}^{n} a_{ij}Y_j\right)$. We then use a classical observation by Valiant (1979), which according to von zur Gathen (1987) even goes back to Hammond (1879): the coefficient of the monomial $Y_1Y_2\dots Y_n$ is $\textsc{per}(A)$. Thus, the coefficient of the monomial $Y_1Y_2\dots Y_n$ in $Q - dY_1Y_2\dots Y_n$ is 0 if and only if $\textsc{per}(A) = d$.

We now show that ZMC for multiplicatively disjoint circuits is in $\mathsf{C_=P}$. This also applies to formulas, since formulas are multiplicatively disjoint. The proof is based on the use of parse trees, which can be seen as objects tracking the formation of monomials during the computation (Malod & Portier 2008) and are the algebraic analog of proof trees (Venkateswaran & Tompa 1989).

Define inductively the parse trees of a circuit C in the following manner:

1. the only parse tree of an input gate is the gate itself,
2. the parse trees of an addition gate α with argument gates β and γ are obtained by taking either a parse tree of β and adding the edge from β to α or by taking a parse tree of γ and adding the edge from γ to α,
3. the parse trees of a multiplication gate α with argument gates β and γ are obtained by taking a parse tree of β and a parse tree of γ and adding the edge from β to α and the edge from γ to α, renaming vertices so that the chosen parse trees of β and γ are disjoint.

The value of a parse tree is defined as the product of the labels of each input gate in the parse tree (note that in the parse tree there may be several copies of a given input gate of the circuit, so that the corresponding label will have as power the number of copies of the gate). It is easy to see that the polynomial computed by a circuit is the sum of the values of its parse trees:

$$C(\bar{x}) = \sum_{T \text{ parse tree of } C} \text{value}(T).$$

In the case of a multiplicatively disjoint circuit, any parse tree is a subgraph of the circuit. In this case, a parse tree can be equivalently seen as a subgraph defined by a subset of T of the edges satisfying the following properties:

1. it contains the output gate,
2. for any addition gate α, if T contains an edge with origin α, then T contains exactly one edge with destination α,
3. for any multiplication gate α, if T contains an edge with origin α, then T contains both (all) edges with destination α,
4. for any gate α, if T contains an edge with destination α, then T contains an edge with origin α.

Consider a multiplicatively disjoint circuit C and a monomial m, where the input gates of C are labeled either by a variable or by -1. A parse tree T contributes to the monomial m in the output polynomial if, when computing the value of the tree, we get exactly the powers in m; this contribution has coefficient $+1$ if the number of gates labeled -1 in T is even, and it has coefficient -1 if this number is odd. The coefficient of m is thus equal to 0 if and only if the number of trees contributing positively is equal to the number of trees contributing negatively.

Let us represent a parse tree by a boolean word $\bar{\epsilon}$, by indicating which edges of C appear in the parse tree (the length N of the words is therefore the number of edges in C). Some of these words will not represent a valid parse tree, but this can be tested in polynomial time. Consider the following language L composed of triples $(C, m, \epsilon_0\bar{\epsilon})$ such that:

1. $\epsilon_0 = 0$ and $\bar{\epsilon}$ encodes a valid parse tree of C which contribute positively to m,

2. or $\epsilon_0 = 1$ and $\bar{\epsilon}$ does not encode a valid parse tree contributing negatively to m.

Then the number of $\bar{\epsilon}$ such that $(C, m, 0\bar{\epsilon})$ belongs to L is the number of parse trees contributing positively to m and the number of $\bar{\epsilon}$ such that $(C, m, 1\bar{\epsilon})$ belongs to L is equal to 2^N minus the number of parse trees contributing negatively to m. Thus, the number of $\epsilon_0\bar{\epsilon}$ such that $(C, m, \epsilon_0\bar{\epsilon}) \in L$ is equal to 2^N if and only if the number of trees contributing positively is equal to the number of trees contributing negatively, if and only if the coefficient of m is equal to 0 in C. Because L is in P, ZMC for multiplicatively disjoint circuits is in $\mathsf{C_{=}P}$. □

THEOREM 3.2. ZMC *belongs to* $\mathsf{coRP}^{\mathsf{PP}}$.

PROOF. Given a circuit C, a monomial m and a prime number p written in binary, COEFFSLP is the problem of computing modulo p the coefficient of the monomial m in the polynomial computed by C. It is shown in Kayal & Saha (2011) that COEFFSLP belongs to $\mathsf{FP}^{\#\mathsf{P}}$.

We now describe a randomized algorithm to decide ZMC. Let c be the constant given in Lemma 2.7. Consider the following algorithm to decide ZMC given a circuit C of size n and a monomial m, using CoeffSLP as an oracle. First choose uniformly at random an integer p smaller than 2^{cn}. If p is not prime, accept. Otherwise, compute the coefficient a of the monomial m in C with the help of the oracle and accept if $a \equiv 0 \mod p$. Since $|a| \leq 2^{2^n}$, Lemma 2.7 ensures that the above is a correct one-sided error probabilistic algorithm for ZMC. This yields ZMC $\in \mathsf{coRP}^{\mathrm{CoeffSLP}}$. Hence, ZMC $\in \mathsf{coRP}^{\mathsf{PP}}$. □

We now consider ZMC on monotone circuits. Notice that in the univariate case, this problem is equivalent to decide if the output of a circuit over sets of natural numbers with operations $\cup$ and $+$ omits a given number. This was shown by McKenzie and Wagner to be coNP-complete for both formulas and circuits. This result can be easily extended to the multivariate case and gives the following.

Theorem 3.3 (McKenzie & Wagner 2007). ZMC *is* coNP*-complete both for monotone formulas and monotone circuits.*

We now give a result linking the ZMC problem to other questions on polynomials computed by circuits. We define the following problem.

GapMonSLP
Input: Univariate arithmetic circuit C over X, $a, b \in \mathbb{N}$.
Problem: Decide if the polynomial computed by C contains no monomial of the form X^c for $a \leq c \leq b$.

GapMonSLP can be seen as a generalization of the degree problem, called DegSLP in Allender *et al.* (2009) (see also Section 6). This generalization can actually be shown to be hard as it has the same complexity as ZMC.

Proposition 3.4. GapMonSLP *is equivalent to* ZMC*.*

Proof. The general case of ZMC easily reduces to ZMC for univariate circuits: we briefly explain the argument below and refer to Allender *et al.* (2009) for further details. Given a circuit C over the variables $X_1, \dots, X_n$ and a monomial $m = X_1^{d_1} \dots X_n^{d_n}$, we can compute a circuit C' over Y and a monomial $m' = Y^d$ such that the coefficient of m' in C' is zero if and only if the coefficient of m in C is zero. Indeed, define C' by substituting each variable X_i with Y^{M^i} in C for $M := 2^{|C|} + 1$ and let $d = \sum_{i=1}^{n} d_i M^i$. The coefficient of $m' = Y^d$ in C' is zero if and only if the coefficient of m in C in zero. Since the univariate case of ZMC is a special case of GapMonSLP, this shows that ZMC reduces to GapMonSLP.

For the other direction, consider (C, a, b) an instance of GapMonSLP over the variable X. Let $C' = C \cdot (1 + XY)^{b-a}$. The circuit C' has polynomial size since $(1 + XY)^b$ can be computed with a circuit of size $O(\log b)$. The coefficient $P(Y)$ of X^b in C' is the zero polynomial if and only if (C, a, b) is a positive instance of GapMonSLP. Now replace Y in C' with $B := 2^{2^{|C'|^2}}$ (obtained by repeated squaring from 2) to obtain a circuit C''. Note that the polynomial P has at most $2^{|C'|}$ monomials, with coefficients bounded by $2^{2^{2|C'|}}$. From the proof of Allender *et al.* (2009, Prop. 2.2), it follows that $P(B) = 0$ if and only if P is the zero polynomial. That is, the coefficient of X^b in C'' is zero if and only if (C, a, b) is a positive instance of GapMonSLP. □

4. Counting monomials

We now turn to the problem of counting the monomials of a polynomial represented by a circuit.

CountMon
Input: Arithmetic circuit C, $d \in \mathbb{N}$.
Problem: Decide if the polynomial computed by C has at least d monomials.

To study the complexity of CountMon, we will look at what we call extending monomials. Given two monomials M and m, we say that M is m-extending if $M = mm'$ and m and m' have no common variable. We start by studying the problem of deciding the existence of an extending monomial.

ExistExtMon
Input: Arithmetic circuit C, monomial m.
Problem: Decide if the polynomial computed by C contains an m-extending monomial.

Proposition 4.1. ExistExtMon *is in* $\mathsf{RP}^{\mathsf{PP}}$. *For multiplicatively disjoint circuits it is* $\mathsf{C}_{\neq}\mathsf{P}$*-complete.*

Proof. We first show the first upper bound. So let (C, m) be an input for ExistExtMon where C is a circuit in the variables $X_1, \ldots, X_n$. Without loss of generality, suppose that $X_1, \ldots, X_r$ are the variables appearing in m. Let $d = 2^{|C|}$: d is a bound on the degree of the polynomial computed by C. We define $C' = \prod_{i=r+1}^{n}(1 + Y_i X_i)^d$ for new variables Y_i. We have that C has an m-extending monomial if and only if in the product CC' the polynomial $P(Y_{r+1}, \ldots, Y_n)$, which is the coefficient of $m \prod_{i=r+1}^{n} X_i^d$, is not identically 0. Observe that P is not given explicitly but can be evaluated modulo a random prime with an oracle for CoeffSLP. Thus, it can be checked if P is identically 0 with the classical Schwartz–Zippel–DeMillo–Lipton lemma (DeMillo & Lipton 1978; Schwartz 1980; Zippel 1979) (see for example Arora & Barak 2009, p. 529). It follows that ExistExt$Mon \in \mathsf{RP}^{\mathsf{PP}}$.

The upper bound in the multiplicatively disjoint setting is easier: we can guess an m-extending monomial M and then output the answer of an oracle for the complement of ZMC, to check whether M appears in the computed polynomial. This establishes containment in $\exists\mathsf{C}_{\neq}\mathsf{P}$ which by Lemma 2.6 is $\mathsf{C}_{\neq}\mathsf{P}$.

For hardness, we reduce to ExistExtMon the $\mathsf{C}_{\neq}\mathsf{P}$-complete problem $\textsc{per}_{\neq}$, i.e., the complement of the $\textsc{per}_{=}$ problem introduced for the proof of Theorem 3.1. We use essentially the same reduction constructing a circuit $Q := \prod_{i=1}^{n} \left(\sum_{j=1}^{n} a_{ij} Y_j\right)$. Observe that the only potential extension of $m := Y_1 Y_2 \ldots Y_n$ is m itself and has the coefficient $\textsc{per}(A)$. Thus, $Q - dY_1 Y_2 \ldots Y_n$ has an m-extension if and only if $\textsc{per}(A) \neq d$. □

CountExtMon
Input: Arithmetic circuit C, $d \in \mathbb{N}$, monomial m.
Problem: Decide if the polynomial computed by C has at least d m-extending monomials.

Proposition 4.2. CountExtMon *is* $\mathsf{PP}^{\mathsf{PP}}$*-complete.*

Proof. Clearly, CountExtMon belongs to $\mathsf{PP}^{\mathrm{ZMC}}$, and thus, with Theorem 3.2, it is in $\mathsf{PP}^{\mathsf{coRP}^{\mathsf{PP}}}$. Using Lemma 2.8, we get membership in $\mathsf{PP}^{\mathsf{PP}}$. To show hardness, we reduce the problem $\mathsf{CC}_{\neq}$3SAT to CountExtMon.

$\mathsf{CC}_{\neq}$3SAT
Input: 3SAT-formula $F(\bar{x}, \bar{y})$, $k, \ell \in \mathbb{N}$.
Problem: Decide if there are at least k assignments to $\bar{x}$ such that there are not exactly ℓ assignments to $\bar{y}$ such that F is satisfied.

Because the construction of the proof of Cook's Theorem is parsimonious, the problem $\mathsf{CC}_{\neq}$3SAT is $\mathsf{CC}_{\neq}\mathsf{P}$-complete. With Lemma 2.3, the hardness of CountExtMon for $\mathsf{PP}^{\mathsf{PP}}$ follows.

Let $(F(\bar{x}, \bar{y}), k, \ell)$ be an instance for the problem $\mathsf{CC}_{\neq}$3SAT. Without loss of generality, we may assume that $\bar{x} = (x_1, \ldots, x_n)$ and $\bar{y} = (y_1, \ldots, y_n)$ and that no clause contains a variable in both negated and non-negated form. Let $\Gamma_1, \ldots, \Gamma_s$ be the clauses of F.

For each literal u of the variables in $\bar{x}$ and $\bar{y}$, we define a monomial $I(u)$ in the variables $X_1, \ldots, X_n, Z_1, \ldots, Z_s$ in the following way:

$$I(x_i) = X_i \prod_{\{j \mid x_i \in \Gamma_j\}} Z_j, \qquad I(\neg x_i) = \prod_{\{j \mid \neg x_i \in \Gamma_j\}} Z_j,$$

$$I(y_i) = \prod_{\{j \mid y_i \in \Gamma_j\}} Z_j, \qquad I(\neg y_i) = \prod_{\{j \mid \neg y_i \in \Gamma_j\}} Z_j.$$

From these monomials, we compute a formula C by

$$C := \prod_{i=1}^{n} \left(I(x_i) + I(\neg x_i)\right) \prod_{i=1}^{n} \left(I(y_i) + I(\neg y_i)\right). \tag{4.3}$$

We fix a mapping *mon* from the assignments of F to the monomials computed by C: Let $\bar{\alpha}$ be an assignment to $\bar{x}$ and $\bar{\beta}$ be an assignment to $\bar{y}$. We define $mon(\bar{\alpha}\bar{\beta})$ as the monomial obtained in the expansion of C by choosing the following terms. If $\alpha_i = 0$, choose $I(\neg x_i)$, otherwise choose $I(x_i)$. Similarly, if $\beta_i = 0$, choose $I(\neg y_i)$, otherwise choose $I(y_i)$.

The monomial $mon(\bar{\alpha}\bar{\beta})$ has the form $\prod_{i=1}^{n} X_i^{\alpha_i} \prod_{j=1}^{s} Z_j^{\gamma_j}$, in which γ_j is the number of true literals in Γ_j under the assignment $\bar{\alpha}\bar{\beta}$. Then F is true under $\bar{\alpha}\bar{\beta}$ if and only if $mon(\bar{\alpha}\bar{\beta})$ has the monomial $\prod_{j=1}^{s} Z_j$. Thus, F is true under $\bar{\alpha}\bar{\beta}$ if and only if $mon(\bar{\alpha}\bar{\beta}) \prod_{j=1}^{s} \left(1 + Z_j + Z_j^2\right)$ has the factor $\prod_{i=1}^{n} X_i^{\alpha_i} \prod_{j=1}^{s} Z_j^3$.

We set $C' = C \prod_{j=1}^{s} \left(1 + Z_j + Z_j^2\right)$. Consider an assignment $\bar{\alpha}$ to $\bar{x}$. The coefficient of the monomial $\prod_{i=1}^{n} X_i^{\alpha_i} \prod_{j=1}^{s} Z_j^3$ in C' is the number of assignments $\bar{\beta}$ such that $\bar{\alpha}\bar{\beta}$ satisfies F. Thus, we get

$$\begin{aligned}
& (F(\bar{x}, \bar{y}), k, \ell) \in \mathsf{CC}_{\neq}\text{3SAT} \\
\Leftrightarrow\ & \text{there are at least } k \text{ assignments } \bar{\alpha} \text{ to } \bar{x} \text{ such that the} \\
& \quad \text{monomial } \prod_{i=1}^{n} X_i^{\alpha_i} \prod_{j=1}^{s} Z_j^3 \text{ does not have coefficient} \\
& \quad \ell \text{ in } C' \\
\Leftrightarrow\ & \text{there are at least } k \text{ assignments } \bar{\alpha} \text{ to } \bar{x} \text{ such that the} \\
& \text{monomial } \prod_{i=1}^{n} X_i^{\alpha_i} \prod_{j=1}^{s} Z_j^3 \text{ occurs in} \\
& C'' := C' - \ell \prod_{i=1}^{n} (1 + X_i) \prod_{j=1}^{s} Z_j^3 \\
\Leftrightarrow\ & \text{there are at least } k \text{ tuples } \bar{\alpha} \text{ such that } C'' \text{ contains} \\
& \text{the monomial } \prod_{i=1}^{n} X_i^{\alpha_i} \prod_{j=1}^{s} Z_j^3 \\
\Leftrightarrow\ & C'' \text{ has at least } k\ \Big(\prod_{j=1}^{s} Z_j^3\Big)\text{-extending monomials.}
\end{aligned}$$

□

THEOREM 4.4. COUNTMON *is* $\mathsf{PP}^{\mathsf{PP}}$*-complete. It is* $\mathsf{PP}^{\mathsf{PP}}$*-hard even for unbounded fan-in formulas of depth* 4.

Proof. CountMon can be easily reduced to CountExtMon since the number of monomials of a polynomial is the number of 1-extending monomials. Therefore, CountMon belongs to $\mathsf{PP}^{\mathsf{PP}}$.

To show hardness, it is enough to prove that instances of CountExtMon constructed in Proposition 4.2 can be reduced to CountMon in logarithmic space. The idea of the proof is that we make sure that the polynomial for which we count all monomials contains all monomials that are not m-extending. Thus, we know how many non-m-extending monomials it contains and we can compute the number of m-extending monomials from the number of all monomials. We could use the same strategy to show in general that CountExtMon reduces to CountMon, but by considering the instance obtained in the proof of Proposition 4.2 and analyzing the extra calculations below, we get hardness for unbounded fanin formulas of depth 4.

So let (C'', k, m) be the instance of CountExtMon constructed in the proof of Proposition 4.2, with $m = \prod_{j=1}^{c} Z_j^3$. We therefore need to count the monomials computed by C'' which are of the form $f(X_1, \ldots, X_n) \prod_{j=1}^{c} Z_j^3$. The circuit C'' is multilinear in X, and the Z_j can only appear with powers in $\{0, 1, 2, 3, 4, 5\}$. So the non-m-extending monomials computed by C'' are all products of a multilinear monomial in the X_i and a monomial in the Z_j where at least one Z_j has a power in $\{0, 1, 2, 4, 5\}$. Fix j, then all monomials that are not m-extending because of Z_j are computed by the formula

$$\tilde{C}_j := \Big(\prod_{i=1}^{n}(X_i+1)\Big)\Big(\prod_{j' \neq j}\sum_{p=0}^{5} Z_{j'}^{p}\Big)\left(1 + Z_j + Z_j^2 + Z_j^4 + Z_j^5\right). \tag{4.5}$$

Thus, the formula $\tilde{C} := \sum_j \tilde{C}_j$ computes all non-m-extending monomials that C'' can compute. The coefficients of monomials in C'' cannot be smaller than $-\ell$ where ℓ is part of the instance of $\mathsf{CC}_{\neq}$3SAT from which we constructed (C'', k, m) before. So the formula $C^* := C'' + (\ell+1)\tilde{C}$ contains all non-m-extending monomials that C'' can compute and it contains the same extending monomials. There are $2^n 6^c$ monomials of the form that C'' can compute, only 2^n of which are m-extending, which means that there are $2^n(6^c - 1)$ monomials computed by C^* that are not m-extending.

As a consequence, C'' has at least k m-extending monomials if and only if C^* has at least $2^n(6^c - 1) + k$ monomials. □

THEOREM 4.6. COUNTMON *is* $\mathsf{PP}^{\mathsf{NP}}$*-complete both for monotone formulas and monotone circuits.*

PROOF. We first show hardness for monotone formulas. The argument is very similar to the proof of Theorem 4.4. Consider the following canonical $\mathsf{C}\exists\mathsf{P}$-complete problem $\mathsf{C}\exists$3SAT.

$\mathsf{C}\exists$3SAT
Input: 3SAT-formula $F(\bar{x}, \bar{y})$, $k \in \mathbb{N}$.
Problem: Decide if there are at least k assignments $\bar{\alpha}$ to $\bar{x}$ such that $F(\bar{\alpha}, \bar{y})$ is satisfiable.

We reduce $\mathsf{C}\exists$3SAT to COUNTMON. With Lemma 2.2 the hardness for $\mathsf{PP}^{\mathsf{NP}}$ follows. Consider a 3SAT-formula $F(\bar{x}, \bar{y})$. Let $n = |\bar{x}| = |\bar{y}|$ and let c be the number of clauses of F. Define the polynomial $C^* = C + \sum_{j=1}^{c} \tilde{C}_j$ where C is defined by (4.3) and $\tilde{C}_j$ by (4.5). The analysis is similar to the proof of Theorem 4.4. The polynomial C^* is computed by a monotone arithmetic formula and has at least $2^n(6^c - 1) + k$ monomials if and only if (F, k) is a positive instance of $\mathsf{C}\exists$3SAT.

We now prove the upper bound. Recall that COUNTMON $\in \mathsf{PP}^{\mathrm{ZMC}}$. From Theorem 3.3, it follows that COUNTMON on monotone circuits belongs to $\mathsf{PP}^{\mathsf{NP}}$. □

5. Multilinearity

In this section, we consider the effect of multilinearity on our problems. We will not consider promise problems, and therefore, the multilinear variants of our problems must first check if the computed polynomial is multilinear. We start by showing that this step is not difficult, indeed, it is equivalent to the problem ACIT.

ACIT
Input: Arithmetic circuit C.
Problem: Decide if the polynomial computed by C is the zero polynomial.

CheckML
Input: Arithmetic circuit C.
Problem: Decide if the polynomial computed by C is multilinear.

Proposition 5.1. CheckML *is equivalent to* ACIT.

Proof. Reducing ACIT to CheckML is easy: Simply multiply the input with X^2 for an arbitrary variable X. The resulting circuit is multilinear if and only if the original circuit was 0.

For the other direction, the idea is to compute the second derivatives of the polynomial computed by the input circuit and check if they are 0.

So let C be a circuit in the variables $X_1, \ldots, X_n$ that is to be checked for multilinearity. For each i, we inductively compute a circuit C_i that computes the second derivative with respect to X_i. To do so, for each gate v in C the circuit, C_i has three gates v_i, v_i' and v_i''. The polynomial computed by v in C will be computed by v_i in C_i, its first derivative with respect to X_i will be computed by v_i' and its second by v_i''. For the input gates, the construction is obvious. If v is a $+$-gate with children u and w, we have $v_i = u_i + w_i$, $v_i' = u_i' + w_i'$ and $v_i'' = u_i'' + w_i''$. If v is a $\times$-gate with children u and w we have $v_i = u_i w_i$, $v_i' = u_i' w_i + u_i w_i'$ and $v_i'' = u_i'' w_i + 2u_i' w_i' + u_i w_i''$. It is easy to see that the constructed circuit computes indeed the second derivative with respect to X_i.

Next, we compute $C' := \sum_{i=1}^{n} Y_i C_i$ for new variables Y_i. We have that C' is identically zero if and only if C is multilinear. Also, C' can easily be constructed in logarithmic space. □

Next, we show that the problem gets much harder if, instead of asking whether *all* the monomials in the polynomial computed by a circuit are multilinear, we ask whether at least *one* of the monomials is multilinear.

MonML
Input: Arithmetic circuit C.
Problem: Decide if the polynomial computed by C contains a multilinear monomial.

The problem MonML lies at the heart of fast exact algorithms for deciding k-paths by Koutis (2008) and Williams (2009) (although in these papers the polynomials are in characteristic 2 which changes the problem a little). This motivated Chen & Fu (2010, 2011) to consider monML, show that it is #P-hard and give algorithms for the bounded depth version. We provide further information on the complexity of this problem.

Proposition 5.2. MonML *is in* $\mathsf{RP}^{\mathsf{PP}}$. *It is* $\mathsf{C}_{\neq}\mathsf{P}$*-complete for multiplicatively disjoint circuits.*

Proof. To prove the first upper bound, consider an input circuit C in variables $X_1, \ldots, X_n$. We set $C' = \prod_{i=1}^{n}(1 + X_iY_i)$. Then C computes a multilinear monomial if and only if in the product CC' the coefficient polynomial $P(Y_1, \ldots, Y_n)$ of $\prod_{i=1}^{n} X_i$ is not identically 0. This can be tested as in the proof of Proposition 4.1, thus establishing MonML $\in \mathsf{RP}^{\mathsf{PP}}$.

The $\mathsf{C}_{\neq}\mathsf{P}$-completeness in the multiplicatively disjoint case can be proved in the same way as in Proposition 4.1. □

We now turn to our first problem, namely deciding whether a monomial appears in the polynomial computed by a circuit, in the multilinear setting.

ML-ZMC
Input: Arithmetic circuit C, monomial m.
Problem: Decide if C computes a multilinear polynomial in which the monomial m has coefficient 0.

Proposition 5.3. ML-ZMC *is equivalent to* ACIT.

Proof. We first show that ACIT reduces to ML-ZMC. So let C be an input for ACIT. Allender *et al.* (2009) have shown that ACIT reduces to a restricted version of ACIT in which all inputs are -1, and thus, the circuit computes a constant. Let C_1 be the result of this reduction. Then C computes identically 0 if and only if the constant coefficient of C_1 is 0. This establishes the first direction.

For the other direction, let (C, m) be the input, where C is an arithmetic circuit and m is a monomial. First check if m is multilinear, if not output 1 or any other nonzero polynomial. Next we construct a circuit C_1 that computes the homogeneous component of degree $\deg(m)$ of C with the classical method (see for example Bürgisser 2000, Lemma 2.14). Observe that if C computes a multilinear polynomial, so does C_1. We now plug in 1 for the variables that appear in m and 0 for all other variables, call the resulting (constant) circuit C_2. If C_1 computes a multilinear polynomial, then C_2 is zero if and only if m has coefficient 0 in C_1. The end result of the reduction is $C^* := C_2 + ZC_3$ where Z is a new variable and C_3 is a circuit which is identically 0 iff C computes a multilinear polynomial (obtained via Proposition 5.1). C computes a multilinear polynomial and does not contain the monomial m if and only if both C_2 and ZC_3 are identically 0, which happens if and only if their sum is identically 0. □

In the case of our second problem, counting the number of monomials, the complexity falls to PP.

ML-CountMon
Input: Arithmetic circuit C, $d \in \mathbb{N}$.
Problem: Decide if the polynomial computed by C is multilinear and has at least d monomials.

Proposition 5.4. ML-CountMon *is* PP*-complete (for Turing reductions).*

Proof. We first show ML-CountMon $\in \mathsf{PP}$. To do so, we use CheckML to check that the polynomial computed by C is multilinear. Then counting monomials can be done in $\mathsf{PP}^{\text{ML-ZMC}}$, and ML-ZMC is in coRP. By Lemma 2.8 the class $\mathsf{PP}^{\mathsf{coRP}}$ is simply PP.

For hardness, we can reduce the computation of the $\{0, 1\}$-permanent to ML-CountMon. The proposition follows, because the $\{0,1\}$-permanent is #P-complete for Turing reductions. So let A be a 0-1-matrix and $d \in \mathbb{N}$ and we have to decide if $\mathrm{PER}(A) \geq d$. We get a matrix B from A by setting $b_{ij} := a_{ij}X_{ij}$. Because every entry of B is either 0 or a distinct variable, we have that, when we compute the permanent of B, every permutation that yields a nonzero summand yields a unique monomial. This means that there are no cancellations, so that $\mathrm{PER}(A)$ is the number of monomials in $\mathrm{PER}(B)$.

The problem is now that no small circuits for the permanent are known, and thus, $\mathrm{PER}(B)$ is not a good input for ML-CountMon. But because there are no cancellations, we have that $\mathrm{DET}(B)$ and $\mathrm{PER}(B)$ have the same number of monomials. So take a small circuit for the determinant (for instance the one given in Mahajan & Vinay 1997) and substitute its inputs by the entries of B. The result is a circuit C which computes a polynomial whose number of monomials is $\mathrm{PER}(A)$. Observing that the determinant, and thus the polynomial computed by C, is multilinear completes the proof. □

6. Univariate circuits

In this section, we briefly study the case of univariate circuits. One problem related to ZMC is to compute the degree of a polynomial given by an arithmetic circuit. This problem was first introduced in Allender *et al.* (2009) under the name DegSLP.

DegSLP
Input: Arithmetic circuit C, $d \in \mathbb{N}$.
Problem: Decide if the degree of the polynomial computed by C is smaller than d.

Allender *et al.* (2009) also introduced the problem EquSLP, which is the problem ACIT restricted to circuits with no indeterminates (i.e., computing integers).

EquSLP
Input: Arithmetic circuit C computing an integer.
Problem: Decide if the integer computed is 0.

In the general case, Allender et al. remark that EquSLP and ACIT are equivalent and they are known to be in coRP. For DegSLP, the best known upper bound is $\mathsf{coRP}^{\mathsf{PP}}$ (Kayal & Saha 2011) and it is an open problem to obtain a lower bound better than P which was obtained Koiran & Perifel (2007). For ZMC we have given a $\mathsf{coRP}^{\mathsf{PP}}$ upper bound and a $\mathsf{C}_{=}\mathsf{P}$ lower bound. Finally, we have shown that CountMon is $\mathsf{PP}^{\mathsf{PP}}$-complete.

In contrast to these differing complexities, we first show that in the case of univariate multiplicatively disjoint circuits, all these problems have equivalent complexities (the case of CountMon is slightly different and treated after the others).

Proposition 6.1. *In the case of univariate multiplicatively disjoint circuits, the problems* DegSLP, ZMC, EquSLP *and* ACIT *are equivalent under logspace reductions. This holds in the monotone and in the general case.*

Proof. The proof we give works both in the general case and in the monotone case. Clearly, EquSLP is a special case of ACIT, and it can also be decided by asking for the constant coefficient in the problem ZMC, or by asking whether the degree is smaller than 0 in DegSLP.

Conversely, we first remark that given a univariate multiplicatively disjoint circuit C of size s, we can construct a multiplicatively disjoint circuit C' of size $O(s^3)$, with $s+1$ output gates computing the coefficients of the polynomial computed by C (the degree of C cannot be greater than s). This is done by the classical argument for computing the homogeneous components of a circuit, noting that if we start from a multiplicatively disjoint circuit, we get a multiplicatively disjoint circuit. Indeed, for each gate α in C, we have in C' the gates $\alpha_0, \dots, \alpha_s$ computing the relevant coefficients of the polynomial computed by α. Then if α is an addition gate with arguments β and γ the gate α_i in C' is also an addition gate with arguments β_i and γ_i. If α is a multiplication gate with arguments β and γ the gate α_i in C' computes $\sum_{k=0}^{i} \beta_k \gamma_{i-k}$. Each product in this sum multiplies a "β" gate with a "γ" gate, so that multiplicatively disjointness is maintained in the construction.

It is now easy to show the converse reductions. In particular, it directly gives the reduction from ZMC to EquSLP.

To reduce ACIT to EquSLP, we apply the above construction, then square all the coefficients of the polynomial and add them up, so that the resulting integer circuit computes 0 iff the starting circuit computed the 0 polynomial.

To reduce DegSLP to EquSLP, we just need to check wether all coefficients of degree at least d are 0, which can be done in a way similar to the reduction from ACIT to EquSLP. □

We now show that, for univariate multiplicatively disjoint circuits, all the problems considered above are complete for LOGCFL in the monotone case and $\mathsf{C_{=}LOGCFL}$ in the general case. We first recall basic facts about these classes.

LOGCFL is the class of all languages that can be reduced in logarithmic space to a context-free language. We will use the characterization of LOGCFL by logspace-uniform semi-unbounded AC^1 circuits (Venkateswaran 1991), or by logspace-uniform circuits of bounded formal degree, and therefore, as mentioned at the end of the paragraph on circuits, also by logspace-uniform circuits where the AND gates are disjoint (Malod & Portier 2008): this follows from the characterization by semi-unbounded fanin circuits of logarithmic depth, and then duplicating gates to ensure that multiplications are disjoint.

In the non-monotone case, we also need to consider the class $\mathsf{C_{=}LOGCFL}$. Instead of using a machine-based definition as we did in Section 2 for $\mathsf{C_{=}}\mathcal{C}$ when the class $\mathcal{C}$ contains P, it is simpler to define this class by arithmetizing a circuit definition of LOGCFL (Allender 2004). A function $f : \{0,1\}^* \to \mathbb{N}$ is defined to be in $\#\mathsf{LOGCFL}$ if there is a logspace-uniform family of multiplicatively disjoint, monotone arithmetic circuits computing f (Malod & Portier 2008). A function $f : \{0,1\}^* \to \mathbb{Z}$ is defined to be in $\mathsf{GapLOGCFL}$ if and only if it is the difference of two functions in $\#\mathsf{LOGCFL}$. Finally, a language L is defined to be in $\mathsf{C_{=}LOGCFL}$ if and only if there is a function $f \in \mathsf{GapLOGCFL}$ such that $x \in L$ if and only if $f(x) = 0$.

PROPOSITION 6.2. *In the case of monotone univariate multiplicatively disjoint circuits, the problems* DEGSLP, ZMC, EQUSLP, ACIT *and* COUNTMON *are* LOGCFL*-complete.*

PROOF. For all of these problems apart from COUNTMON, it is enough to show that in the monotone case the problem EQUSLP is LOGCFL-complete. Note that a monotone arithmetic circuit counts the number of satisfying parse trees of the corresponding Boolean circuit (Allender 2004) (where a + gate is replaced by and OR and a × gate by an AND): this number is nonzero iff the Boolean circuit evaluates to true. The completeness follows since LOGCFL can also be characterized with circuits with disjoint AND gates (Malod & Portier 2008).

For the upper bound for COUNTMON, given a circuit C and an integer d, we start from the construction given in Proposition 6.1 which yields a family of arithmetic circuits computing the coefficients of the polynomial computed by C. By the remark above, in the monotone case each of these circuits is nonzero iff the associated Boolean circuit is nonzero. We can then add a small Boolean circuit which adds up these Boolean values and compares it with d, staying in LOGCFL.

Finally, the complement of EQUSLP trivially reduces to the problem COUNTMON by asking whether the number of monomials is at least 1, so that COUNTMON is LOGCFL-hard. □

PROPOSITION 6.3. *In the case of univariate multiplicatively disjoint circuits,* DEGSLP, ZMC, EQUSLP, ACIT *are* $\mathsf{C_=LOGCFL}$*-complete.*

PROOF. Once again, we just need to consider EQUSLP, and the result is then direct from the definition given above for $\mathsf{C_=LOGCFL}$. □

PROPOSITION 6.4. *In the case of univariate multiplicatively disjoint circuits,* COUNTMON *is* $\mathsf{C_=LOGCFL}$*-hard and is in* $\mathsf{L}^{\mathsf{C_=LOGCFL}}$.

PROOF. The hardness follows from the argument given at the end of the proof of Proposition 6.2. The upper bound is clear using Proposition 6.3, since we only need to add up a small number of answers to ZMC and then compare with d. □

7. Conclusion

In this paper, we have strengthened the known connection between the counting hierarchy and arithmetic circuits by showing that natural questions on arithmetic circuits are complete for different classes in CH. We consider it as likely that other questions on arithmetic circuits could be shown to be connected to CH with similar techniques.

Since the preliminary version of this paper (Fournier *et al.* 2012) was published, several problems from this paper have been considered by Mahajan *et al.* (2012) for very restricted circuit classes, so-called read-once/twice formulas and branching programs. For these classes, the complexity of our problems often but not always drops considerably.

Let us also remark that the techniques from this paper have found an application in a recent paper by Mittmann *et al.* (2012): The notion of degeneracy considered there, to which algebraic independence in positive characteristic can be reduced, is shown to be hard by reduction from ZMC. It would be interesting to see if a similar hardness result can be shown for algebraic independence itself.

Let us close the paper with some open questions: The $\mathsf{C_=P}$ lower bound for ZMC does not match the upper bound of $\mathsf{coRP}^{\mathsf{PP}}$ completely. Can this upper bound be derandomized to show that ZMC is in $\mathsf{C_=P}$ also in the general case?

DegSLP is in our opinion, one of the most puzzling open questions in arithmetic circuit complexity. While it is widely believed to be hard, not even conditional hardness results are known for it. Our contribution to the understanding of DegSLP has been very modest, but we feel that the direction it proposes might be promising. Maybe a better understanding of tractable classes of polynomials computed by restricted classes of circuits will lead to a better understanding of the general problem. So are there any other classes of circuits for which DegSLP is tractable? Are there any multivariate classes?

Acknowledgements

An extended abstract of this paper has appeared as Fournier *et al.* (2012). We would like to thank Sylvain Perifel for helpful discussions. The results of this paper were conceived while Stefan Mengel was visiting the Équipe de Logique Mathématique at Université Paris Diderot. He would like to thank Arnaud Durand for making this stay possible, thanks to funding from ANR ENUM (ANR-07-BLAN-0327). Stefan Mengel was partially supported by DFG grants BU 1371/2-2 and BU 1371/3-1. He would also like to thank his supervisor Peter Bürgisser for helpful advice.

References

E. Allender (2004). Arithmetic circuits and counting complexity classes. In *Complexity of computations and proofs*, volume 13 of *Quad. Mat.*, 33–72. Dept. Math., Seconda Univ. Napoli, Caserta.

E. Allender, R. Beals & M. Ogihara (1999). The complexity of matrix rank and feasible systems of linear equations. *Computational Complexity* **8**(2), 99–126.

E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen & P. B. Miltersen (2009). On the Complexity of Numerical Analysis. *SIAM J. Comput.* **38**(5), 1987–2006.

E. Allender & K. W. Wagner (1993). Counting hierarchies: polynomial time and constant depth circuits. *Current trends in theoretical computer science: essays and Tutorials* **40**, 469.

S. Arora & B. Barak (2009). *Computational complexity: a modern approach*. Cambridge University Press.

P. Bürgisser (2000). *Completeness and reduction in algebraic complexity theory*, volume 7. Springer Verlag.

P. Bürgisser (2009). On defining integers and proving arithmetic circuit lower bounds. *Computational Complexity* **18**(1), 81–103.

Zhixiang Chen & Bin Fu (2010). Approximating Multilinear Monomial Coefficients and Maximum Multilinear Monomials in Multivariate Polynomials. In *COCOA (1)*, 309–323.

ZHIXIANG CHEN & BIN FU (2011). The Complexity of Testing Monomials in Multivariate Polynomials. In *COCOA*, 1–15.

RICHARD A. DEMILLO & RICHARD J. LIPTON (1978). A Probabilistic Remark on Algebraic Program Testing. *Inf. Process. Lett.* **7**(4), 193–195. URL http://dblp.uni-trier.de/db/journals/ipl/ipl7.html#DemilloL78.

H. FOURNIER, G. MALOD & S. MENGEL (2012). Monomials in arithmetic circuits: Complete problems in the counting hierarchy. In *STACS*, 362–373.

S. GARG & E. SCHOST (2009). Interpolation of polynomials given by straight-line programs. *Theor. Comput. Sci.* **410**(27-29), 2659–2662.

J. VON ZUR GATHEN (1987). Feasible arithmetic computations: Valiant's hypothesis. *Journal of Symbolic Computation* **4**(2), 137–172.

F. GREEN (1993). On the Power of Deterministic Reductions to $C_=P$. *Theory of Computing Systems* **26**(2), 215–233.

J. HAMMOND (1879). Question 6001. *Educ. Times* **32**, 179.

J. HEINTZ & C. P. SCHNORR (1980). Testing polynomials which are easy to compute (Extended Abstract). In *Proceedings of the twelfth annual ACM symposium on Theory of computing*, STOC '80, 262–272. ACM, New York, NY, USA. ISBN 0-89791-017-6. URL http://doi.acm.org/10.1145/800141.804674.

L. A. HEMASPAANDRA & M. OGIHARA (2002). *The complexity theory companion.* Springer Verlag.

M. JANSEN & R. SANTHANAM (2011). Permanent Does Not Have Succinct Polynomial Size Arithmetic Circuits of Constant Depth. In *ICALP*, 724–735.

V. KABANETS & R. IMPAGLIAZZO (2004). Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity* **13**, 1–46. ISSN 1016-3328. URL http://dx.doi.org/10.1007/s00037-004-0182-6.

N. KAYAL & C. SAHA (2011). On the Sum of Square Roots of Polynomials and related problems. In *IEEE Conference on Computational Complexity*, 292–299.

Adam R. Klivans & Daniel Spielman (2001). Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, STOC '01, 216–223. ACM, New York, NY, USA. ISBN 1-58113-349-9. URL http://doi.acm.org/10.1145/380752.380801.

P. Koiran & S. Perifel (2007). The complexity of two problems on arithmetic circuits. *Theor. Comput. Sci.* **389**(1-2), 172–181.

P. Koiran & S. Perifel (2011). Interpolation in Valiant's theory. *Computational Complexity* 1–20.

I. Koutis (2008). Faster Algebraic Algorithms for Path and Packing Problems. In *ICALP*, 575–586.

J. Kwisthout, H. L. Bodlaender & L. C. van der Gaag (2011). The Complexity of Finding *k*th Most Probable Explanations in Probabilistic Networks. In *SOFSEM*, 356–367.

M. Mahajan, B. V. Raghavendra Rao & K. Sreenivasaiah (2012). Identity Testing, Multilinearity Testing, and Monomials in Read-Once/Twice Formulas and Branching Programs. In *MFCS*, 655–667.

M. Mahajan & V. Vinay (1997). A combinatorial algorithm for the determinant. In *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, 730–738. Society for Industrial and Applied Mathematics.

G. Malod & N. Portier (2008). Characterizing Valiant's algebraic complexity classes. *J. Complexity* **24**(1), 16–38.

P. McKenzie & K. W. Wagner (2007). The complexity of membership problems for circuits over sets of natural numbers. *Computational Complexity* **16**(3), 211–244.

J. Mittmann, N. Saxena & P. Scheiblechner (2012). Algebraic Independence in Positive Characteristic – A p-Adic Calculus. *ArXiv e-prints* .

M. Mundhenk, J. Goldsmith, C. Lusena & E. Allender (2000). Complexity of Finite-Horizon Markov Decision Process Problems. *Journal of the ACM (JACM)* **47**(4), 681–720.

A. Schönhage (1979). On the Power of Random Access Machines. In *ICALP*, 520–529.

J. T. Schwartz (1980). Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM* **27**(4), 701–717. ISSN 0004-5411. URL http://doi.acm.org/10.1145/322217.322225.

Yann Strozecki (2013). On Enumerating Monomials and Other Combinatorial Structures by Polynomial Interpolation. *Theory Comput. Syst.* **53**(4), 532–568.

J. Torán (1988). Succinct Representations of Counting Problems. In *AAECC*, 415–426.

J. Torán (1991). Complexity Classes Defined by Counting Quantifiers. *J. ACM* **38**(3), 753–774.

L.G. Valiant (1979). Completeness classes in algebra. In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, 249–261. ACM.

H. Venkateswaran (1991). Properties that Characterize LOGCFL. *J. Comput. Syst. Sci.* **43**(2), 380–404.

H. Venkateswaran & M. Tompa (1989). A New Pebble Game that Characterizes Parallel Complexity Classes. *SIAM J. Comput.* **18**(3), 533–549.

K. W. Wagner (1986). The Complexity of Combinatorial Problems with Succinct Input Representation. *Acta Informatica* **23**(3), 325–356.

R. Williams (2009). Finding paths of length k in $O^*(2^k)$ time. *Information Processing Letters* **109**(6), 315–318.

Richard Zippel (1979). Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, EUROSAM '79, 216–226. Springer-Verlag, London, UK, UK. ISBN 3-540-09519-5. URL http://dl.acm.org/citation.cfm?id=646670.698972.

Manuscript received 27 February 2013

Hervé Fournier
Institut de Mathématiques de
Jussieu, UMR 7586 CNRS,
Univ Paris Diderot,
Sorbonne Paris Cité,
75205 Paris, France.
fournier@math.
univ-paris-diderot.fr

Guillaume Malod
Institut de Mathématiques de
Jussieu, UMR 7586 CNRS,
University Paris Diderot,
Sorbonne Paris Cité,
75205 Paris, France.
malod@math.
univ-paris-diderot.fr

Stefan Mengel
LIX,
École Polytechnique,
Palaiseau, France.
mengel@lix.polytechnique.fr