The Complexity of two Problems on Arithmetic Circuits
Pascal Koiran, Sylvain Perifel

To cite this version:
Pascal Koiran, Sylvain Perifel. The Complexity of two Problems on Arithmetic Circuits. 2007. ensl-00167613
The Complexity of two Problems on Arithmetic Circuits

Pascal Koiran and Sylvain Perifel
LIP*, École Normale Supérieure de Lyon. 
24th August 2007

Abstract By using arithmetic circuits, encoding multivariate polynomials may be drastically more efficient than writing down the list of monomials. Via the study of two examples, we show however that such an encoding can be hard to handle with a Turing machine even if the degree of the polynomial is low. Namely we show that deciding whether the coefficient of a given monomial is zero is hard for P^{#P} under strong nondeterministic Turing reductions. As a result, this problem does not belong to the polynomial hierarchy unless this hierarchy collapses. For polynomials over fields of characteristic $k > 0$, this problem is Mod_k P-complete. This gives a coNP^{Mod_k P} algorithm for deciding an upper bound on the degree of a polynomial given by a circuit in fields of characteristic $k > 0$.

1 Introduction

Multivariate polynomials are intensively used in computer algebra. The naive way of encoding such polynomials is to write down the list of the coefficients of all monomials, but even low-degree polynomials may have an exponentially large number of such coefficients (consider for example the determinant). That is why this possibility is not always suitable.

Arithmetic circuits can sometimes remedy this situation since their size is in general much smaller than the list of all coefficients. An arithmetic circuit over a field $K$ is a circuit that contains addition and multiplication gates over $K$ (instead of OR and AND gates for usual boolean circuits). With such gates, and with inputs consisting of variables and constants of $K$, the circuit computes a polynomial (see the preliminaries below for details). For example, the polynomial $(x+y)^2$ can be computed by a circuit of size $O(n)$ by “repeated squaring”, whereas the number of monomials is exponentially large.

* UMR 5668 ENS Lyon, CNRS, UCBL, INRIA.
Though compact, this manner of storing data is not always easy to
handle. There are efficient randomized algorithms for testing whether a
polynomial is identically zero (see e.g. [14]), for example, but many other
problems become hard when dealing with arithmetic circuits. One way to
simplify them is to consider only circuits of polynomial “formal degree”
([10] or [12], the degree of the polynomial if no cancellation would occur).
In this direction, Erich Kaltofen [7] designed efficient algorithms for the
greatest common divisor of two polynomials, for instance.

This paper, in the same spirit as [1], investigates two natural prob-
lems concerning multivariate polynomials given by arithmetic circuits. We
are concerned with the complexity of these problems in the usual Turing
machine model. One of them is the computation of the degree of a poly-
nomial (Section 5). This problem is efficiently solved by a probabilistic
algorithm in the case where the “formal degree” is polynomially bounded;
unfortunately, in characteristic 0, the upper bound of [1] in the counting
hierarchy is not improved. However, for polynomials over fields of char-
teristic \( k > 0 \), we obtain the better bound \( \text{coNP}^{\text{Mod}_k} \). We shall also
study the problem of deciding whether the coefficient of a given monomial
is zero (Sections 3 and 4). We show that even on polynomially-bounded
formal degree instances, this problem is hard, that is, hard for \( \text{P}^\# \text{P} \) under
strong nondeterministic Turing reductions. We also provide a complete
characterization of the complexity of this problem when the field is of
positive characteristic.

Throughout this paper, we heavily rely on the results of [10]. Though
stated in an algebraic context in [10] (namely Valiant’s framework), we
show indeed that they can also be useful for proving results in boolean
complexity related to the manipulation of polynomials given by arithmetic
circuits.

2 Preliminaries

An arithmetic circuit over a field \( K \) is a directed acyclic graph, each of the
vertices is of indegree 0 or 2, one and only one vertex being of outdegree
0. This last vertex is called output, and vertices of indegree zero are called
inputs. An input is labeled by a variable \( x_i \) or a constant among \( \{-1, 0, 1\} \)
in \( K \). Every other vertex is called a gate and is labeled by \(+\) or \( \times \). The
polynomial computed by a vertex is defined inductively as follows:

- the polynomial computed by an input is its label (i.e. a constant or a
  variable);
- the polynomial computed by a + gate is the sum of its entries;
– the polynomial computed by a $\times$ gate is the product of its entries.

The polynomial computed by a circuit is the polynomial computed by the output of the circuit. Note that the only allowed constants of the field are $-1, 0$ and $1$, so the coefficients of our polynomials are in $\mathbb{Z}$ (characteristic zero) or in $\mathbb{Z}_k$ (characteristic $k > 0$). The size of a circuit $C$, written $|C|$, is the number of its vertices. See Figure 1 below for an example where cancellations occur.

![Figure 1](image)

**Figure 1.** A circuit of size 16 computing the polynomial $x_1^4 + x_2$.

Another useful quantity concerning arithmetic circuits is their formal degree. The formal degree of a vertex in an arithmetic circuit is inductively defined as follows:

– the formal degree of an input is 1;
– the formal degree of a $+$ gate is the maximum of the formal degrees of its entries;
– the formal degree of a $\times$ gate is the sum of the formal degrees of its entries.

The formal degree of an arithmetic circuit is the formal degree of its output. We denote the formal degree of a circuit $C$ by $\deg_f(C)$. Note that one can easily compute the formal degree of a circuit by induction.

Here we are interested in the complexity of handling arithmetic circuits by (usual, boolean) Turing machines. Note that an arithmetic circuit is easily encodable in binary since it merely consists of a labeled directed graph. Hence, such circuits can be manipulated by Turing machines. We shall study two problems: computing the degree of a polynomial, and deciding whether the coefficient of a monomial is zero. We shall define two versions of both problems, a “bounded” version where a part of the input
is given in unary, and the unbounded one where the whole input is in binary.

The first language we shall consider concerns the degree of the polynomial. If $C$ is an arithmetic circuit, the degree of the (multivariate) polynomial computed by $C$ is denoted by $\deg(C)$. The language $\text{DEG}$ is defined as follows:

$$\text{DEG} = \{(C, d) : \deg(C) \leq d\},$$

where $C$ is an arithmetic circuit and $d$ an integer. A “bounded” version of $\text{DEG}$ is also defined:

$$\text{DEG}_b = \{(C, 1^d) : \deg(C) \leq d\},$$

that is, $d$ is given in unary, hence the size of the input is $\geq d$.

The second language we shall study concerns the coefficient of a monomial in a polynomial. A monomial $m$ is encoded by the list of the exponents of all the variables in binary. If we denote by $\text{coef}_C(m)$ the coefficient of the monomial $m$ in the polynomial computed by $C$, we define

$$\text{ZMC} = \{(C, m) : \text{coef}_C(m) = 0\}$$

($\text{ZMC}$ stands for Zero Monomial Coefficient), and its bounded version:

$$\text{ZMC}_b = \{(C, m, 1^d) : \text{coef}_C(m) = 0 \text{ and } \deg_f(C) = d\}.$$

As above, $C$ is an arithmetic circuit, and $m$ a monomial.

Note that these languages only depend on the characteristic of the underlying field, because the only allowed constants in the arithmetic circuits are $-1$, $0$ and $1$. When dealing with fields of characteristic $k > 0$, we will use the superscript $k$ as in $\text{ZMC}^k$, $\text{ZMC}_b^k$, $\text{DEG}^k$ and $\text{DEG}_b^k$.

We finally briefly recall a few complexity classes; we refer the reader to [13] and [5] for instance for further details. If $L$ is a language, we denote by $\text{P}^L$ (resp. $\text{NP}^L$) the class of languages recognized by a deterministic (resp. nondeterministic) polynomial time Turing machine with oracle $L$. If $C$ is a set of languages, $\text{P}^C$ (resp. $\text{NP}^C$) denotes $\cup_{L \in C} \text{P}^L$ (resp. $\cup_{L \in C} \text{NP}^L$).

We define inductively the classes $\Sigma^p_i$ as follows: $\Sigma^p_0 = \text{P}$ and $\Sigma^p_{i+1} = \text{NP}^{\Sigma^p_i}$. The polynomial hierarchy is $\text{PH} = \cup_{i \geq 0} \Sigma^p_i$.

We shall also encounter probabilistic classes. The class $\text{coRP}$ is the set of languages $L$ recognized by a probabilistic Turing machine that doesn’t make any mistake on inputs in $L$, and has a small probability of error for inputs outside of $L$. More precisely, a language $L$ is in $\text{coRP}$ if there exists $A \in \text{P}$ and a polynomial $q(n)$ such that for all $x \in \{0, 1\}^*$:
– if \( x \in L \) then \( \text{Card}\{y \in \{0,1\}^{|x|} : (x,y) \in A\} = 0 \);  
– if \( x \notin L \) then \( \text{Card}\{y \in \{0,1\}^{|x|} : (x,y) \in A\} \geq 2^{|x|}-1 \).

BPP is analogous but both-sided errors are allowed:  
– if \( x \in L \) then \( \text{Card}\{y \in \{0,1\}^{|x|} : (x,y) \in A\} \geq 3/4 \cdot 2^{|x|} \);  
– if \( x \notin L \) then \( \text{Card}\{y \in \{0,1\}^{|x|} : (x,y) \in A\} < 1/4 \cdot 2^{|x|} \).

Finally, PP is similar but the threshold is 1/2 and is the same for both accepting and rejecting conditions. The class PP gives rise to the counting hierarchy: \( \text{CH} = \text{P} \cup \text{PP} \cup \text{PPPP} \cup \ldots \).

Let us now define “counting classes”. The class \#P is the set of functions that count the number of accepting paths of a non-deterministic Turing machine working in polynomial time. More precisely, \( f : \{0,1\}^* \to \mathbb{N} \) is in \#P if there exists \( A \in \text{P} \) and a polynomial \( q(n) \) such that  
\[
f(x) = \text{Card}\{y \in \{0,1\}^{|x|} : (x,y) \in A\}.
\]

The closure of \#P under subtraction is called GapP: thus, a function GapP takes its values in \( \mathbb{Z} \) and is the difference of two \#P functions.

An analogous definition provides counting classes modulo \( k \): if \( k \) is an integer greater than 1, Mod\(_k\)P is the set of languages \( L \) such that  
\[
x \in L \iff \text{Card}\{y \in \{0,1\}^{|x|} : (x,y) \in A\} \not\equiv 0 \mod k.
\]

If \( k \) is prime, Mod\(_k\)P is closed under complement \( \Box \), so we can take \( \equiv 0 \) in the definition instead of \( \not\equiv 0 \).

Let us recall a few inclusions:  
\[
\text{P} \subseteq \text{NP} \subseteq \text{PH} \subseteq \text{P}^\#P \subseteq \text{CH}, \quad \text{and} \quad \text{P} \subseteq \text{coRP} \subseteq \text{BPP} \subseteq \Sigma^p_2.
\]

We shall also make use of strong nondeterministic Turing reductions, introduced by Long \( \Box \). This is defined as follows: \( B \leq_{sn}^T A \) iff \( B \in (\text{NP}^A \cap \text{coNP}^A) \).

The next section studies ZMC (and its bounded version): we first show that over fields of characteristic zero, ZMC\(_b\) \( \in \text{P}^\#P \), ZMC \( \in \text{CH} \) and these languages are outside PH unless it collapses. Then we study fields of characteristic \( k > 0 \), in which ZMC\(_k\) and ZMC\(_b^k\) are Mod\(_k\)P-complete. Finally, Section \( \Box \) deals with DEG and DEG\(_b\).

3 The coefficient of a monomial in characteristic zero

Here we work on polynomials with coefficients in \( \mathbb{Z} \), i.e. we do not use modular arithmetic. The following two results are shown.
Proposition 1. ZMC_b is \( \text{P}^\# \text{P} \)-complete under strong nondeterministic Turing reductions \( \leq_T^{\text{sn}} \).

Proposition 2. ZMC is in CH and is \( \text{P}^\# \text{P} \)-hard under strong nondeterministic Turing reductions \( \leq_T^{\text{sn}} \).

Remark that the hardness result of Proposition 2 is implied by that of Proposition 1. The latter is shown in Section 3.1. The upper bounds of both propositions are proved in Section 3.2.

3.1 Hardness for \( \text{P}^\# \text{P} \)

The language Perm We define the language Perm as a decision problem for the permanent. We recall that the permanent of a matrix \( M = (m_{i,j})_{1 \leq i,j \leq n} \) is defined as

\[
\text{per}(M) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} m_{i,\sigma(i)}
\]

where \( S_n \) is the set of all permutations of \( \{1, \ldots, n\} \). Despite its similarity with the determinant, the permanent seems to be hard to compute (even for 0-1 matrices) since it is \( \# \text{P} \)-complete under counting reductions [8].

The language Perm is the following:

\[
\text{Perm} = \{(M, v) : \text{per}(M) = v\},
\]

where \( M \) is a 0-1 square matrix, and \( v \) an integer (this language also appears in [8]). We show that Perm is \( \text{P}^\# \text{P} \)-hard under strong nondeterministic Turing reductions, i.e.

\[
\text{P}^\# \text{P} \subseteq \text{NP}^{\text{Perm}} \cap \text{coNP}^{\text{Perm}}.
\]

Let \( L \in \text{P}^\# \text{P} \). Since computing the permanent of a 0-1 matrix is \( \# \text{P} \)-complete, there exists a polynomial-time deterministic oracle Turing machine \( M \) that decides \( L \) with the help of the oracle per. It is easy to simulate this machine with a nondeterministic one, call it \( \mathcal{N} \), that consults the oracle Perm. We proceed as follows. While \( M \) does not use its oracle, \( \mathcal{N} \) simply simulates \( M \) in a deterministic manner. As soon as \( M \) asks for the value of a permanent, say \( \text{per}(M) \) for a matrix \( M \), the machine \( \mathcal{N} \) guesses the value of this permanent and check for a correct guess thanks to its oracle Perm: only the correct path (corresponding to the true value of the permanent) is continued, whereas the other paths are rejected.
The Complexity of two Problems on Arithmetic Circuits

Our machine $N$ thus exactly has one path that behaves like $M$, the other being rejecting paths. So $N$ recognizes $L$, and we have shown the inclusion $\text{P#P} \subseteq \text{NP}^\text{Perm}$. The other inclusion $\text{P#P} \subseteq \text{coNP}^\text{Perm}$ holds by closure of $\text{P#P}$ under complement.

**Remark 1.** Toda’s theorem ([15]) asserts that $\text{PH} \subseteq \text{P#P}$. It yields $\text{PH} \subseteq \text{NP}^\text{Perm}$.

This implies that Perm does not lie in the polynomial hierarchy unless it collapses. Indeed, if $\text{Perm} \in \Sigma_i^p$ for some $i$, then $\text{PH} \subseteq \text{NP}^{\Sigma_i^p} = \Sigma_{i+1}^p$, and PH collapses to the $(i + 1)$-th level.

**ZMC_b is harder than Perm**

**Remark 2.** The following ideas already appear in [15]. We nevertheless detail the reduction here for the sake of completeness.

We now show that $\text{ZMC}_b$ is harder than Perm. For, if $M = (m_{i,j})$ is a matrix ($1 \leq i, j \leq n$) and $v$ an integer, we use the following classical multivariate polynomial, already appearing in e.g. [17] (without the term $-vX_1 \cdots X_n$, though):

$$P(M, v)(X_1, \ldots, X_n) = \prod_{j=1}^{n} (\sum_{i=1}^{n} m_{i,j}X_i) - vX_1 \cdots X_n.$$  

We claim that the coefficient of the monomial $X_1 \cdots X_n$ in this polynomial is $\text{per}(M) - v$. Indeed, consider the product

$$\left( \sum_{i=1}^{n} m_{1,i}X_i \right) \left( \sum_{i=1}^{n} m_{2,i}X_i \right) \cdots \left( \sum_{i=1}^{n} m_{n,i}X_i \right).$$

In order to develop it, in each factor one has to choose one term of the sum. Since we are concerned with the coefficient of $X_1 \cdots X_n$, one has to choose different variables $X_i$ for each factor. Therefore, for each permutation $\sigma$ of $\{1, \ldots, n\}$, the product yields a term of the form $(\prod_i m_{i,\sigma(i)})X_1 \cdots X_n$, thus the coefficient of $X_1 \cdots X_n$ in the product is

$$\sum_{\sigma} \prod_i m_{i,\sigma(i)} = \text{per}(M).$$

Now, taking the term $-vX_1 \cdots X_n$ into account, we obtain the announced coefficient $\text{per}(M) - v$. 

ZMC_b is harder than Perm

**Remark 2.** The following ideas already appear in [15]. We nevertheless detail the reduction here for the sake of completeness.

We now show that ZMC_b is harder than Perm. For, if $M = (m_{i,j})$ is a matrix ($1 \leq i, j \leq n$) and $v$ an integer, we use the following classical multivariate polynomial, already appearing in e.g. [17] (without the term $-vX_1 \cdots X_n$, though):

$$P(M, v)(X_1, \ldots, X_n) = \prod_{j=1}^{n} (\sum_{i=1}^{n} m_{i,j}X_i) - vX_1 \cdots X_n.$$  

We claim that the coefficient of the monomial $X_1 \cdots X_n$ in this polynomial is $\text{per}(M) - v$. Indeed, consider the product

$$\left( \sum_{i=1}^{n} m_{1,i}X_i \right) \left( \sum_{i=1}^{n} m_{2,i}X_i \right) \cdots \left( \sum_{i=1}^{n} m_{n,i}X_i \right).$$

In order to develop it, in each factor one has to choose one term of the sum. Since we are concerned with the coefficient of $X_1 \cdots X_n$, one has to choose different variables $X_i$ for each factor. Therefore, for each permutation $\sigma$ of $\{1, \ldots, n\}$, the product yields a term of the form $(\prod_i m_{i,\sigma(i)})X_1 \cdots X_n$, thus the coefficient of $X_1 \cdots X_n$ in the product is

$$\sum_{\sigma} \prod_i m_{i,\sigma(i)} = \text{per}(M).$$

Now, taking the term $-vX_1 \cdots X_n$ into account, we obtain the announced coefficient $\text{per}(M) - v$. 

ZMC_b is harder than Perm

**Remark 2.** The following ideas already appear in [15]. We nevertheless detail the reduction here for the sake of completeness.

We now show that ZMC_b is harder than Perm. For, if $M = (m_{i,j})$ is a matrix ($1 \leq i, j \leq n$) and $v$ an integer, we use the following classical multivariate polynomial, already appearing in e.g. [17] (without the term $-vX_1 \cdots X_n$, though):

$$P(M, v)(X_1, \ldots, X_n) = \prod_{j=1}^{n} (\sum_{i=1}^{n} m_{i,j}X_i) - vX_1 \cdots X_n.$$  

We claim that the coefficient of the monomial $X_1 \cdots X_n$ in this polynomial is $\text{per}(M) - v$. Indeed, consider the product

$$\left( \sum_{i=1}^{n} m_{1,i}X_i \right) \left( \sum_{i=1}^{n} m_{2,i}X_i \right) \cdots \left( \sum_{i=1}^{n} m_{n,i}X_i \right).$$

In order to develop it, in each factor one has to choose one term of the sum. Since we are concerned with the coefficient of $X_1 \cdots X_n$, one has to choose different variables $X_i$ for each factor. Therefore, for each permutation $\sigma$ of $\{1, \ldots, n\}$, the product yields a term of the form $(\prod_i m_{i,\sigma(i)})X_1 \cdots X_n$, thus the coefficient of $X_1 \cdots X_n$ in the product is

$$\sum_{\sigma} \prod_i m_{i,\sigma(i)} = \text{per}(M).$$

Now, taking the term $-vX_1 \cdots X_n$ into account, we obtain the announced coefficient $\text{per}(M) - v$. 

ZMC_b is harder than Perm

**Remark 2.** The following ideas already appear in [15]. We nevertheless detail the reduction here for the sake of completeness.

We now show that ZMC_b is harder than Perm. For, if $M = (m_{i,j})$ is a matrix ($1 \leq i, j \leq n$) and $v$ an integer, we use the following classical multivariate polynomial, already appearing in e.g. [17] (without the term $-vX_1 \cdots X_n$, though):

$$P(M, v)(X_1, \ldots, X_n) = \prod_{j=1}^{n} (\sum_{i=1}^{n} m_{i,j}X_i) - vX_1 \cdots X_n.$$  

We claim that the coefficient of the monomial $X_1 \cdots X_n$ in this polynomial is $\text{per}(M) - v$. Indeed, consider the product

$$\left( \sum_{i=1}^{n} m_{1,i}X_i \right) \left( \sum_{i=1}^{n} m_{2,i}X_i \right) \cdots \left( \sum_{i=1}^{n} m_{n,i}X_i \right).$$

In order to develop it, in each factor one has to choose one term of the sum. Since we are concerned with the coefficient of $X_1 \cdots X_n$, one has to choose different variables $X_i$ for each factor. Therefore, for each permutation $\sigma$ of $\{1, \ldots, n\}$, the product yields a term of the form $(\prod_i m_{i,\sigma(i)})X_1 \cdots X_n$, thus the coefficient of $X_1 \cdots X_n$ in the product is

$$\sum_{\sigma} \prod_i m_{i,\sigma(i)} = \text{per}(M).$$

Now, taking the term $-vX_1 \cdots X_n$ into account, we obtain the announced coefficient $\text{per}(M) - v$. 

ZMC_b is harder than Perm
From the input \((M, v)\), it is easy to build in polynomial time an arithmetic circuit with polynomially-bounded formal degree\(^1\) that computes \(P_{(M,v)}\). Then the language ZMC\(_b\) enables us to decide Perm by asking whether the coefficient of \(X_1 \cdots X_n\) in \(P_{(M,v)}\) is zero. We conclude that ZMC\(_b\) is harder than Perm for polynomial-time many-one reductions. This shows the hardness results of Propositions 1 and 2. This also implies that ZMC\(_b\) (and a fortiori ZMC) is not in the polynomial hierarchy unless it collapses.

### 3.2 The help of a \#P-oracle

We now show that ZMC\(_b\) is in P\#P. We need the following definition of [10, section 5.2] (also in [11]).

**Definition 1.** Let \(\alpha\) be a gate whose inputs are \(\beta\) and \(\gamma\). The gate \(\alpha\) is disjoint if the subcircuits associated to \(\beta\) and \(\gamma\) are disjoint.

A circuit is said to be multiplicatively disjoint if all its multiplication gates are disjoint.

Multiplicatively disjoint circuits are useful in the light of the following result of Malod and Portier [11, Lemma 2] (also in [10]).

**Lemma 1.** Let \(C\) be a circuit of size \(s\) and formal degree \(d\). There exists a multiplicatively disjoint circuit \(C'\) of size \(\leq ds\) which computes the same polynomial as \(C\). Furthermore, constructing \(C'\) from \(C\) requires only time polynomial in \(|C|\) and \(d\).

In order to show that ZMC\(_b\) is in P\#P, we rely on the proof of Theorem 3 of [11, section 2.4]: the idea is that the coefficient of the monomial \(m\) in a multiplicatively disjoint circuit is merely the number of possible “developments” that lead to \(m\). By a “development” we mean a tree which represents the choices in the distribution of \(\times\) over \(+\). More precisely, if \(C\) is a multiplicatively disjoint arithmetic circuit, a development \(D\) is a subgraph of \(C\) satisfying the following properties:

- every \(\times\) gate in \(D\) has both inputs in \(D\);
- every \(+\) gate in \(D\) has exactly one input in \(D\);
- \(D\) contains the output gate of the circuit, and every gate in \(D\) different from the output gate has at least one child in \(D\).

\(^1\) An immediate construction provides a formal degree of roughly \(2n\) for the product, and \(n \log n\) for \(v X_1 \cdots X_n\) (since \(|v| \leq n!\), because it is a permanent of a 0-1 \(n \times n\) matrix). The overall circuit thus has formal degree \(\leq 2n(1 + \log n) \leq |C|^2\).
The monomial computed by a development is merely the product of its variables. If the development contains the constant 0, then the development will be called neutral. Otherwise, if the development contains the constant −1, then it will be called negative, else positive. The following result of [10] justifies the introduction of the notion of developments.

**Lemma 2.** Let $C$ be a multiplicatively disjoint circuit and $m$ a monomial $X_1^{a_1} \cdots X_n^{a_n}$. Let $d^+(m)$ and $d^-(m)$ be the numbers of positive and negative developments computing $m$, respectively. Then the coefficient of $m$ in the polynomial computed by $C$ is equal to $d^+(m) - d^-(m)$.

We finally have the following result.

**Lemma 3.** The function $f$ which, given a multiplicatively disjoint circuit $C$ and a monomial $m$, computes the coefficient of $m$, is in GapP.

**Proof.** Checking whether a tree is a development corresponding to a given monomial is in P. Define $A^+(m)$ (respectively $A^-(m)$) to be the set of positive (resp. negative) developments of $C$ computing $m$. The coefficient of $m$ is then $|A^+(m)| - |A^-(m)|$ and is computable in GapP because deciding $A^+(m)$ and $A^-(m)$ takes only polynomial time. \(\Box\)

For the bounded version ZMC$_b$, the formal degree of the circuit is bounded by the size of the input since it is given in unary. Therefore one can compute in polynomial time an equivalent multiplicatively disjoint circuit by Lemma 1. By Lemma 3, computing the coefficient of $m$ and testing it to zero (i.e., deciding ZMC$_b$) is done in P$\#P$. This concludes the proof of Proposition 1.

In fact, we obtain a polynomial-time algorithm with only one call to a GapP oracle $f$. This can be converted into one call to a #P oracle as follows. Let $f^+$ and $f^-$ be the #P-functions such that $f = f^+ - f^-$ and take a polynomial $p(n)$ such that $f^+(x), f^-(x) < 2^{p(|x|)}$ for all $x$. Then the function $g$ defined by $g(x) = f^+(x) + 2^{p(|x|)}f^-(x)$ is in #P and we can recover $f$ from $g$. We have therefore proved a slightly stronger statement, since only one call to the #P oracle is necessary: ZMC$_b \in$ P$\#P[1]$.

**Unbounded version.** However, this construction does not work for the unbounded version, since we cannot transform the original circuit into a multiplicatively disjoint one in polynomial time. We only show here that ZMC is in the counting hierarchy CH.

As in [10], we first build a “generic” arithmetic circuit, called $G_n$, whose inputs can be specialized to simulate any other circuit of size $\leq n$. The Complexity of two Problems on Arithmetic Circuits
The construction proceeds as follows. We define
\[ G_{-n} = 1, G_{-n-1} = x_1, \ldots, G_0^n = x_n. \]
We then compute every possible sum and product of the previous levels, by adding new variables \( a_{l+1,i} \) and \( b_{l+1,i} \):
\[ G_{l+1}^n = \left( \sum_{i=-n}^{l} a_{l+1,i} G_i^n \right) \left( \sum_{i=-n}^{l} b_{l+1,i} G_i^n \right). \]

We then write \( G_n(y_1, \ldots, y_p) = G_n^n \). The circuit \( G_n \) is generic in the sense that the computation of any circuit of size \( \leq n \) can be obtained from \( G_n(y_1, \ldots, y_p) \) by specializing some variables: here is the precise statement coming from \( \text{[10, Prop. 3]} \).

**Lemma 4.** Let \( f(x_1, \ldots, x_k) \) be a polynomial computed by an arithmetic circuit \( C \) of size \( \leq n \). Then

1. there exist \( a_1, \ldots, a_p \) among \( \{-1, 0, 1, 2, x_1, \ldots, x_k\} \) such that \( f(x_1, \ldots, x_k) = G_n(a_1, \ldots, a_p) \);
2. the values of \( a_1, \ldots, a_p \) can be computed from \( C \) in polynomial time.

If \( C \) is a fixed circuit and \( f \) the polynomial computed by \( C \), let \( \tau \) be the substitution of the variables \( y_1, \ldots, y_p \) of \( G_n \) given by the preceding lemma, that is, the map defined by \( \tau(y_i) = a_i \). We will write \( C = \tau(G_n) \).

If \( m' \) is a monomial in \( G_n \), the image \( \tau(m') \) of \( m' \) by the substitution \( \tau \) is the product of a (possibly zero) coefficient and a monomial of \( f \); we will denote by \( \alpha(m') \) the coefficient and by \( \tau'(m') \) the monomial, that is, \( \tau(m') = \alpha(m') \tau'(m') \) where \( \alpha(m') \) is an integer. Note that both \( \tau' \) and \( \alpha \) are computable in polynomial time.

We thus have:
\[ \text{coef}_C(m) = \sum_{m' | \tau'(m') = m} \alpha(m') \text{coef}_{G_n}(m'). \]

Malod has explicitely evaluated the coefficients of \( G_n \) in \( \text{[10, section 5.2.3]} \) by means of binomial coefficients. More precisely, he shows the following result.

**Proposition 3.** The coefficient in \( G_n \) of the monomial
\[ m = \prod_{-n \leq i,j \leq n} a_{i,j}^{\alpha_{i,j}} \prod_{-n \leq i,j \leq n} b_{i,j}^{\beta_{i,j}} \prod_{1 \leq i \leq n} x_i^{\gamma_i} \]
is 0 if \( \gamma_i \neq \sum_{j=-n}^{n} (\alpha_{i-n,j} + \beta_{i-n,j}) \) for some \( i \geq 1 \), and
\[
\prod_{i=1}^{n} \prod_{j=-n}^{i-1} \left( \sum_{k=-n}^{j} \alpha_{i,j} \right) \left( \sum_{k=-n}^{j} \beta_{i,j} \right) \text{ otherwise.}
\]
Thus, if we are given a circuit \( C \), we compute the substitution \( \tau \) such that \( C = \tau(G_n) \) thanks to Lemma 4 and we have:
\[
\text{coef}_C(m) = \sum_{m', \tau(m') = m} \alpha(m') \text{coef}_{G_n}(m').
\]
Bürgisser shows in [4, Corollary 3.8] that the so-called falling factorials \( N(N-1) \ldots (N-k) \) are computable bit by bit in CH on input size \( \log N \), as well as integer divisions and sums and products exponential in the size of the input (Theorem 3.7). Therefore this sum of products of binomial coefficients is computable bit by bit in the counting hierarchy CH. Now, a coNP computation is enough for testing whether all bits of the coefficients are zero. Therefore \( ZMC \in \text{coNP}^{\text{CH}} = \text{CH} \). This concludes the proof of Proposition 2.

Remark 3. That \( ZMC \in \text{CH} \) could also be proven thanks to the result of [1] that the problem BitSLP is in the counting hierarchy. The problem BitSLP consists in computing the bits of an integer computed by an arithmetic circuit. Indeed, similarly as in the proof of Proposition 2.1 of [1], one can substitute to the variables \( x_i \) of our polynomial some integers growing sufficiently fast, for instance \( 2^{2s^2} \), where \( s \) is the size of the circuit. Then, when computing the bits of the integer thus computed, there will be no overlap of the coefficients of the polynomial (because they are bounded by \( 2^{2^2} \) in absolute value). We will therefore be able to recover them in CH thanks to BitSLP.

4 The coefficient of a monomial in positive characteristic

Now we are working over a field of positive characteristic \( k > 0 \), that is, the polynomials computed by our circuits now have coefficients in \( \mathbb{Z}_k \), so we can perform modular arithmetic. This will help a little, since we shall show the following completeness result.

Proposition 4. \( ZMC^k \) and \( ZMC^k_0 \) for fields of characteristic \( k > 0 \) are \( \text{Mod}_k \text{P-complete} \).
Remark 4. Todai’s theorem shows that $\text{PH} \subseteq \text{BPP}^{\text{Mod}_2^P}$. This result is
generalized in [16] to any $k \geq 2$, that is

$$\text{PH} \subseteq \text{BPP}^{\text{Mod}_k^P},$$

which implies that $\text{ZMC}^k$ is not in the polynomial hierarchy unless it
collapses.

4.1 Algorithm

We use the generic polynomial $G_n$ of the preceding section and the same
notations. Given a circuit $C$, one can compute in polynomial time the
substitution $\tau$ such that $C = \tau(G_n)$. Now, showing that the coefficients
of $G_n$ modulo $k$ can be computed in polynomial time is enough in order
to show that the coefficients of $C$ can be computed in $\text{Mod}_k^P$, since the
coefficient of $m$ is expressed as a sum of those of $G_n$:

$$\text{coef}_C(m) = \sum_{m', \tau(m') = m} \alpha(m') \text{coef}_{G_n}(m')$$

and $\text{Mod}_k^P$ is closed under exponential sum if the index of the summation
is computable in polynomial time (as is the case here).

It remains to see why the coefficients of $G_n$ are computable in polyno-
mial time modulo $k$. The main tool here is the following result of Édouard
Lucas [9], thanks to which we can efficiently compute binomial coefficients
modulo $k$.

**Proposition 5 (Lucas, 1878).** Let $k$ be a prime integer. Let $N = \sum_{i=0}^n n_i k^i$ and $M = \sum_{i=0}^m m_i k^i$ be two integers written in base $k$. We have

$$\binom{N}{M} \equiv \prod_{i=0}^{n} \binom{n_i}{m_i} \mod k.$$

Together with Proposition 3, this proves that the computation of the
coefficients of $G_n$ modulo $k$ is done in polynomial time. The remark above
on the closure of $\text{Mod}_k^P$ under exponential sum shows that $\text{ZMC}^k \in \text{Mod}_k^P$: this is the first part of Proposition 4.

4.2 Lower bound

Let us now show that $\text{ZMC}^k_b$ is hard for $\text{Mod}_k^P$. We use a reduction
from “#Hamilton Cycle”, the function that counts the number of Hamil-
ton cycles in a graph. This function is $\#P$-complete under parsimonious
reductions, that is, under counting reductions that preserve the number of solutions, see [13] (one can also use the proof of [13, Th. 18.2] of \#P-completeness of \"\#Hamilton Path\" and then use an easy parsimonious reduction from \"\#Hamilton Path\" to \"\#Hamilton Cycle\"). So it remains complete modulo $k$, i.e. \"\#Hamilton Cycle modulo $k\" is Mod$_k$P-complete. The number of Hamilton cycles in a graph of adjacency matrix $(x_{i,j})_{1 \leq i,j \leq n}$ is given by the polynomial

$$HC_n(x_{i,j}) = \sum_{\sigma} x_{i,\sigma(i)} \prod_{i=1}^{n} x_{i,\sigma(i)},$$

where the sum is taken over all $n$-cycles of $S_n$.

Now, as in the case of the permanent, the number of Hamilton cycles is the coefficient of a monomial in a polynomial $P$ that is obtained in polynomial time from the adjacency matrix of the graph:

**Lemma 5 ([10], Lemma 14).** Let the polynomials $T_{p,i,j}$ be defined as $T_{1,i,j} = x_{i,j} y_{i} z_{j}$ and $T_{p+1,i,j} = \sum_{k=1}^{n} T_{p,i,k} T_{1,k,j}$. Then

- the polynomial $P = T_{n,1,1}$ can be computed by an arithmetic circuit of size $O(n^4)$ and of formal degree $O(n)$;
- the circuit itself is computable in time polynomial in $n$;
- the coefficient of the monomial $y_1 z_1 \cdots y_n z_n$ in $P$ is $HC_n$.

Since $HC_n$ is the coefficient of a monomial in a polynomial $P$ computed by an arithmetic circuit $C$ of polynomial size and polynomial formal degree, the same argument as in the case of the permanent applies: ZMC$^k_b$ is as hard as counting the number of Hamilton cycles modulo $k$, i.e. ZMC$^k_b$ is Mod$_k$P-hard. This concludes the proof of Proposition 4.

5 Computation of the degree

We now turn to the study of the complexity of DEG and DEG$_b$.

5.1 Unbounded version

For the unbounded version, we do not improve the PP$^{PP^{PP^{PP}}}$ upper bound of [4] for fields of characteristic zero. Over a field of positive characteristic, however, Section 4 helps us finding the better bound coNP$^\text{Mod}_k$P. We first show that DEG is P-hard for logspace reductions.
Lower bound. We show that the “Circuit value problem” (CVP) is easier than DEG. CVP is the language consisting of those boolean circuits whose output is TRUE. This is a P-complete problem under logspace reductions \[13\], Th. 8.1. In fact, we can encode a boolean circuit in an arithmetic one, by simulating the boolean operations by polynomials: 
\((x \text{ AND } y) = xy\), \((x \text{ OR } y) = x + y - xy\), and \((\text{NOT } x) = 1 - x\). From the boolean circuit \(B\), we construct the arithmetic circuit \(C_0\) that simulates \(B\); then we multiply the output gate of \(C_0\) by a new variable \(x\), and add 1. Call the new circuit \(C\). The polynomial computed by \(C\) is 1 if the value of \(B\) is FALSE, and \(x + 1\) otherwise. Thus by computing the degree of \(C\), we know the value of \(B\).

Since this procedure uses only logarithmic work space, we conclude that DEG is P-hard for logspace reductions.

Upper bound. Obviously, both problems DEG and ZMC are related. In fact, we have:
\[\deg(C) \leq d \text{ iff } (\text{every monomial of degree } > d \text{ has zero for coefficient}).\]

Since the number of monomials is simply exponential in the size of the circuit, this condition is checkable in coNP with the help of the oracle ZMC. It yields:
\[\text{DEG} \in \text{coNP}^{ZMC}.\]

In characteristic zero, ZMC is in CH, so
\[\text{DEG} \in \text{coNP}^{CH} = \text{CH}.\]

This does not improve the result of \[1\] that DEG is in the counting hierarchy. In characteristic \(k > 0\), however, ZMC is in Mod\(_k\)P, so DEG is in coNP\(^{\text{Mod}_k\text{P}}\). Thus we have proved the following result.

**Proposition 6.** The problem DEG\(^k\) for fields of characteristic \(k > 0\) is P-hard for logspace reductions and is in coNP\(^{\text{Mod}_k\text{P}}\).

5.2 Bounded version

The case of DEG\(_0\) is easy and uses classical techniques. Let \(C\) be a circuit and \(f\) the polynomial it computes: we want to know whether the degree of \(f\) is \(\leq d\). Since \(d\) is given in unary, one can first build in polynomial time an arithmetic circuit computing all the homogeneous components \(f_i\) of degree \(\leq d\) of \(f\), as in \[3\], Prop. 5.28 or \[1\], Lemma 2. We now simply
have to test whether \( f = \sum_{i=0}^{d} f_i \). Testing the equality of two polynomials given by arithmetic circuits is standard by testing the equality at random points, see [14] for example. By working in extensions of the prime field if necessary, it is easy to see that the technique also works in fields of positive characteristic.

As a whole, it yields the following result.

**Proposition 7.** The problem \( \text{DEG}_b \) is in \( \text{coRP} \).

### 6 Summary and further research

Here is a summary of our results. The symbol \( \leq_T^{sn} \) means “strong non-deterministic Turing reduction”, see Section 2. The upper bound for \( \text{DEG} \) in the counting hierarchy is from [1].

<table>
<thead>
<tr>
<th>Characteristic zero</th>
<th>Characteristic ( k &gt; 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZMC(_b)</td>
<td>P(^#P) ( \text{(under } \leq_T^{sn} \text{)} )</td>
</tr>
<tr>
<td>ZMC</td>
<td>CH</td>
</tr>
<tr>
<td>DEG(_b)</td>
<td>?</td>
</tr>
<tr>
<td>DEG</td>
<td>P</td>
</tr>
<tr>
<td></td>
<td>P</td>
</tr>
</tbody>
</table>

There is still a lot of work to do, namely to reduce the gap between the above results. Possible next steps would be to completely characterize ZMC in characteristic zero, and to find a better upper bound than \( \text{P}^{\text{PP}^{\text{PP} \#P}} \) for \( \text{DEG} \).

The authors thank Erich Kaltofen for the suggestion of encoding \( d \) in unary in the language \( \text{DEG}_b \), as well as the anonymous referees for useful remarks.

### References


