**DIMACS Technical Report 97-81**
**December 1997**

# On Arithmetic Branching Programs

by

Amos Beimel[1] Anna Gál[2]

[1]Division of Engineering and Applied Sciences, Harvard University, ESL, 40 Oxford st., Cambridge, MA 02138. Email: beimel@deas.harvard.edu. Supported by grants ONR-M00014-96-1-0550 and ARO-DAAL03-92-G0115. Part of this work was done while the author was a postdoctoral fellow at DIMACS.
[2]Dept. of Computer Sciences, The University of Texas at Austin, Email: panni@cs.utexas.edu. Part of this work was done while the author was a postdoctoral fellow at DIMACS and the Dept. of Computer Science of Princeton University.

---

DIMACS is a partnership of Rutgers University, Princeton University, AT&T Labs-Research, Bell Labs and Bellcore.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

## ABSTRACT

We consider the model of arithmetic branching programs, which is a generalization of modular branching programs. We show that, up to a polynomial factor in size, arithmetic branching programs are equivalent to complements of dependency programs, a model introduced by Pudlák and Sgall [14]. Using this equivalence we prove that dependency programs are closed under conjunction, answering an open problem of [14]. Furthermore, we show that span programs, an algebraic model of computation introduced by Karchmer and Wigderson [10], are at least as strong as arithmetic programs; every arithmetic program can be simulated by a span program of size not more than twice the size of the arithmetic program. Using the above results we give a new proof that $\mathcal{NL}/\text{poly} \subseteq \oplus\mathcal{L}/\text{poly}$, first proved by Wigderson [19]. Our simulation of $\mathcal{NL}/\text{poly}$ is more efficient, and it holds for logspace counting classes over every field.

# 1 Introduction

Algebraic models of computation have received much attention (e.g. [18, 1]). One fundamental question is the power of basic algebraic operations. For example, if we can compute the rank of a matrix then we can check if the rows of the matrix are dependent. Can we compute the rank of a matrix using an oracle for dependency? Since both computations can be performed in polynomial time, this question should be addressed in weaker complexity models. In this work we study algebraic models which supply an arena to study such questions.

We focus on the model of arithmetic branching programs which is a generalization of modular branching programs. For the clarity of the presentation, we first describe the model of modular branching programs and then describe arithmetic branching programs. A mod-$p$ branching program is a directed acyclic graph with two distinguished vertices $s$ and $t$ and edges labeled by literals. It accepts an assignment if the number of paths between $s$ and $t$ that are consistent with the assignment is not equal to 0 mod-$p$ (a path is consistent with an assignment if the assignment satisfies all the labels of the edges in the path). This model corresponds to nondeterministic logspace Turing machines that count the number of accepting paths. Modular branching programs have been defined and studied in [7, 10], and were used to construct private simultaneous protocols in [9]. Modular branching programs are also referred to in the literature as counting branching programs.

An arithmetic branching program (over a field $\mathcal{K}$) is a branching program where every edge in the graph has a weight which is a number taken from the field $\mathcal{K}$. The weight of a path in the graph is defined as the *product* of the weights of the edges in the path. The arithmetic program accepts an assignment if the *sum* of the weights of the consistent paths between $s$ and $t$ is not equal to 0 (where the product and sum are in $\mathcal{K}$). The *size* of an arithmetic program is the number of edges in the graph. Clearly, a mod-$p$ branching program is an arithmetic program over $\mathrm{GF}(p)$ where the weight of every edge is 1.

We investigate the relations between arithmetic branching programs and two other algebraic models: span programs [10] and dependency programs [14]. Karchmer and Wigderson [10] defined span programs and showed, using results of [7], that over fixed finite fields they are equivalent to modular branching programs. That is, a function has a small mod-$p$ branching program if and only if the function has a small span program over $\mathrm{GF}(p)$. Span programs were considered also in [5, 3, 4, 2]. Pudlák and Sgall [14] defined a similar model called dependency programs. They proved that span programs are at least as strong as dependency programs. It is not known whether span programs are actually stronger. We give precise definitions of the above models in the next section. We denote the size of the smallest span program, dependency program and arithmetic program for $f$ by $\mathrm{SP}_{\mathcal{K}}(f)$, $\mathrm{DP}_{\mathcal{K}}(f)$ and $\mathrm{AP}_{\mathcal{K}}(f)$ respectively. Pudlák and Sgall [14] proved that $(\mathrm{SP}_{\mathcal{K}}(f))^{\frac{1}{2}} \leq \mathrm{DP}_{\mathcal{K}}\left(\overline{f}\right) \leq (\mathrm{AP}_{\mathcal{K}}(f))^2$ (only modular branching programs were considered in [14], however their proof holds for arithmetic programs over arbitrary fields). An easy modification of the arguments in [14]

gives

$$\mathrm{DP}_{\mathcal{K}}\left(\overline{f}\right) \leq 2 \cdot \mathrm{AP}_{\mathcal{K}}(f) \ . \tag{1}$$

It is simple to show that arithmetic programs are closed under conjunction. Our first technical result is showing that arithmetic programs are closed under disjunction. We present two constructions. The first construction applies to fields that are not algebraically closed, and the second construction applies to infinite fields. Since finite fields are not algebraically closed, we prove the closure under disjunction of arithmetic programs over any field.

Our second technical result is that $\mathrm{AP}_{\mathcal{K}}(f) \leq \left(\mathrm{DP}_{\mathcal{K}}\left(\overline{f}\right)\right)^{O(1)}$ over any field $\mathcal{K}$. This result, together with results of [14], implies that arithmetic programs are equivalent to complements of dependency programs (up to a polynomial factor in size). That is, a function $f$ has a small arithmetic program if and only if its complement $\overline{f}$ has a small dependency program. Our proof combines results on the computation of the rank [12, 6] and the closure of arithmetic programs under disjunction. Our results also imply that dependency programs are closed under conjunction, solving an open problem in [14]. Furthermore, we show that span programs can simulate arithmetic programs very efficiently. For every function $f$ it holds that $\mathrm{SP}_{\mathcal{K}}(f) \leq 2 \cdot \mathrm{AP}_{\mathcal{K}}(f)$. This generalizes the result of [10] for programs over GF(2). For other fields only a polynomial relation was known [7, 14].

Our last result shows that, over any field, arithmetic programs can efficiently simulate nondeterministic branching programs. This generalizes a result of Wigderson [19] that modular branching programs over GF(2) (and any fixed finite field) can simulate nondeterministic branching programs (e.g., $\mathcal{NL}/\mathrm{poly} \subseteq \oplus\mathcal{L}/\mathrm{poly}$). The construction of Wigderson [19] directly generalizes to arithmetic programs over arbitrary fields. The efficient simulation of nondeterministic branching programs by arithmetic programs over any field also follows from a recent result of Reinhardt and Allender [16] who proved, using the construction of [19], that $\mathcal{NL}/\mathrm{poly} = \mathcal{UL}/\mathrm{poly}$. However, we give an alternative construction which is more efficient, i.e., the resulting arithmetic program is smaller.

**Remark:** Nisan [13] considered a similar complexity model called algebraic branching programs. These programs compute a function from $R^n$ to $R$ (where $R$ is a ring), while arithmetic branching programs compute functions from $\{0,1\}^n$ to $\{0,1\}$. The motivation of [13] was to prove lower bounds on non-commutative computations.

**Organization:** In Section 2 we define the computational models we consider. In Section 3 we prove that arithmetic programs are closed under conjunction and disjunction. In Section 4 we prove that arithmetic programs and complements of dependency programs are equivalent up to a polynomial increase in size. In Section 5 we show that span programs can simulate arithmetic programs with only doubling the size, and in Section 6 we show that over any field arithmetic programs can simulate nondeterministic branching programs with polynomial increase in the size. Finally, in Section 7 we discuss some open problems. We present some background results the appendix.

# 2 Definitions

In this section we define the three complexity models discussed in this paper: arithmetic branching programs, span programs, and dependency programs. We start with some helpful notations. We use bold letters for vectors (e.g., $\mathbf{f}$), and the $u$-th coordinate of $\mathbf{f}$ is denoted by $f(u)$. The variable $x_i$ is also denoted by $x_i^1$ and the negated variable $\overline{x}_i$ is also denoted by $x_i^0$. For a literal $x_i^\epsilon$ we denote its negation by $x_i^{\overline{\epsilon}}$. For an assignment $u \in \{0,1\}^n$ the $i$-th coordinate of $u$ is denoted by $u_i$. An assignment $u$ satisfies a literal $x_i^\epsilon$ if $u_i = \epsilon$, i.e., if $u_i = 1$ then $u$ satisfies $x_i$ and if $u_i = 0$ then $u$ satisfies $\overline{x}_i$. An assignment always satisfies the constant 1. Finally, the vector $\mathbf{1}$ denotes the vector in which each coordinate is 1 (the number of coordinates of $\mathbf{1}$ would be clear from the context). Throughout the paper, whenever we describe an arithmetic program in a figure, the unmarked edges have weight 1 and are labeled by 1.

**Definition 2.1 (Arithmetic Branching Programs)** *An* arithmetic branching program *(or* arithmetic program*) over a field $\mathcal{K}$ is a quintet $C = \langle G, \mu, w, s, t\rangle$ where $G = (V, E)$ is a directed acyclic graph, $\mu : E \to \{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\} \cup \{1\}$ is a function called the labeling function, $w : E \to \mathcal{K}$ is a function called the weight function and $s$ and $t$ are vertices called the source and target vertices.*

*We next define the function computed by the arithmetic program. To a path in the graph we assign a weight which is the* product *of the weights of the edges in the path. A path is consistent with an assignment $u \in \{0,1\}^n$ if the assignment satisfies all the labels of the edges in the path. We define $C(u)$ as the* sum *of the weights of the paths between $s$ and $t$ that are consistent with $u$ (where the product and sum are in $\mathcal{K}$). The arithmetic program $C$ accepts an assignment $u$ if $C(u) \neq 0$. An arithmetic program* computes *a Boolean function $f$ if it accepts exactly those inputs $u$ where $f(u) = 1$. The* size *of an arithmetic program is the number of edges in $G$ that are labeled by a literal. The size of the smallest arithmetic program over $\mathcal{K}$ that computes $f$ is denoted by* $\mathrm{AP}_{\mathcal{K}}(f)$.

In the definition of the size of an arithmetic branching program we ignore edges labeled by 1. This is the common assumption in the definition of any branching program (e.g. [15, 10]). We can assume, w.l.o.g., that there are no parallel edges labeled by 1 and every vertex touches an edge labeled by a literal, hence the number of edges labeled by 1 is at most the square of the number of edges labeled by a literal. Thus, if we do count constant edges then the size of the minimal arithmetic program is at most squared. Furthermore, constant edges can be eliminated if we replace every edge labeled by 1 by two parallel edges, one labeled by $x_1$ and the other by $\overline{x}_1$. A mod-$p$ branching program [7, 10] is an arithmetic program over $\mathrm{GF}(p)$ where the weight of every edge is 1. The program accepts an assignment if the number of consistent paths is not equal to 0 mod-$p$.[1]

[1] In [10] the accepting criterion is different; the program accepts if the number of consistent paths is equal to 1 mod-$p$.

**Definition 2.2 (Span Programs [10])** *A* span program *over $\mathcal{K}$ is a triplet $\widehat{M} = \langle M, \rho, \mathbf{v} \rangle$, where $M$ is a matrix over $\mathcal{K}$, $\mathbf{v}$ is a non-zero row vector called the target vector (it has the same number of coordinates as the number of columns in $M$), and $\rho$ is a labeling of the rows of $M$ by literals from $\{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$ (every row is labeled by one literal, and the same literal can label many rows).*

*A span program accepts or rejects an input by the following criterion. For every input $u \in \{0,1\}^n$ define the sub-matrix $M_u$ of $M$ consisting of those rows whose labels are satisfied by the assignment $u$. The span program $\widehat{M}$* accepts *$u$ if and only if $\mathbf{v} \in \text{span}(M_u)$, i.e., some linear combination of the rows of $M_u$ gives the vector $\mathbf{v}$. A span program* computes *a Boolean function $f$ if it accepts exactly those inputs $u$ where $f(u) = 1$. The* size *of $\widehat{M}$ is the number of rows in $M$. The size of the smallest span program over $\mathcal{K}$ that computes $f$ is denoted by $\text{SP}_{\mathcal{K}}(f)$.*

The number of columns is not counted as part of the size of a span program. It is always possible to use no more columns than the size of the program (since we may restrict the matrix to a set of linearly independent columns without changing the function that is computed). However, in some of our arguments it would be convenient to construct span programs with a large number of columns. The choice of the fixed vector $\mathbf{v}$ does not effect the size of the span program. It is always possible to replace $\mathbf{v}$ with any non-zero vector by changing the basis of the matrix $M$. The default value for the target vector is $\mathbf{1}$.

**Definition 2.3 (Dependency Programs [14])** *A* dependency program *over $\mathcal{K}$ is a pair $\widehat{M} = \langle M, \rho \rangle$, where $M$ is a matrix over $\mathcal{K}$, and $\rho$ is a labeling of the rows of $M$ by literals from $\{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$.*

*A dependency program accepts an input $u$ if and only if the rows of $M_u$ are linearly dependent. A dependency program* computes *a Boolean function $f$ if it accepts exactly those inputs $u$ where $f(u) = 1$. The* size *of $\widehat{M}$ is the number of rows in $M$. The size of the smallest dependency program over $\mathcal{K}$ that computes $f$ is denoted by $\text{DP}_{\mathcal{K}}(f)$.*

In [14], the size of both dependency programs and span programs is defined as the number of columns in the matrix. Using the different size definition may cause at most a factor of $n$ difference in size, and all of our results would remain valid.

# 3 Closure Properties of Arithmetic Programs

In this section we show that if $f_1$ and $f_2$ have small arithmetic programs then $f_1 \wedge f_2$ and $f_1 \vee f_2$ have small arithmetic programs. While the construction of an arithmetic program for $f_1 \wedge f_2$ is simple, the construction of an arithmetic program for $f_1 \vee f_2$ is more tricky and depends on the underlying field. One construction works over fields that are not algebraically closed (e.g., every finite field, the rationals and the reals), and the other construction works over infinite fields. We start with an arithmetic program for $f_1 \wedge f_2$.

**Lemma 3.1** *Let $\mathcal{K}$ be any field. For arbitrary functions $f_1$ and $f_2$ it holds that* $\mathrm{AP}_{\mathcal{K}}(f_1 \wedge f_2) \leq \mathrm{AP}_{\mathcal{K}}(f_1) + \mathrm{AP}_{\mathcal{K}}(f_2)$.

**Proof:** Simply, connect the arithmetic programs of $f_1$ and $f_2$ sequentially. The arithmetic program is illustrated in Fig. 1. □

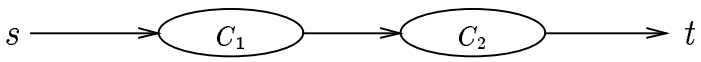


Figure 1: An arithmetic program for $f_1 \wedge f_2$

The arithmetic program for $f_1 \vee f_2$ is more complicated. The first construction works only over fields which are not algebraically closed. We use the following lemma of [8]. We include its proof for completeness.

**Lemma 3.2** *Let $d \geq 2$. Assume that $P(x) \triangleq \sum_{i=0}^{d} a_i x^i$ is an irreducible polynomial over $\mathcal{K}$ of degree exactly $d$ (i.e., $a_d \neq 0$), and $Q(x,y) \triangleq \sum_{i=0}^{d} a_i x^i y^{d-i}$. Then $Q(x,y) = 0$ if and only if $x = 0$ and $y = 0$.*

**Proof:** Clearly if $x = y = 0$ then $Q(x,y) = 0$. Assume $Q(x_0, y_0) = 0$. If $y_0 = 0$ then $x_0 = 0$ and the lemma follows. However, if $y_0 \neq 0$ then $Q(x_0, y_0) = y_0^d \cdot P\left(x_0/y_0\right)$ . Thus, $Q(x_0, y_0) = 0$ for $y_0 \neq 0$ implies $P(x_0/y_0) = 0$, contradicting the assumption that $P(x)$ is irreducible. □

For example, the polynomial $x^2 + 1$ is irreducible over the reals and the rationals, and the polynomial $Q(x,y) = x^2 + y^2$ equals 0 if and only if $x = y = 0$.

**Lemma 3.3** *Assume that there is an irreducible polynomial of degree $d$ over $\mathcal{K}$. Then for arbitrary functions $f_1$ and $f_2$ it holds that* $\mathrm{AP}_{\mathcal{K}}(f_1 \vee f_2) \leq d\left(\mathrm{AP}_{\mathcal{K}}(f_1) + \mathrm{AP}_{\mathcal{K}}(f_2)\right)$ .

**Proof:** Let $C_1$ and $C_2$ be arithmetic programs computing the functions $f_1$ and $f_2$ respectively. In Fig. 2, we describe an arithmetic program $C$ such that $C(u)$ – the sum of the weights of the paths consistent with $u$ in $C$ – equals $Q(C_1(u), C_2(u))$ (where $Q$ is defined as in Lemma 3.2). The arithmetic program $C$ rejects $u$ if $C(u) = 0$, i.e., $Q(C_1(u), C_2(u)) = 0$. By Lemma 3.2, $Q(C_1(u), C_2(u)) = 0$ if and only if $C_1(u) = 0$ and $C_2(u) = 0$, that is, both $C_1$ and $C_2$ reject $u$, and $(f_1 \vee f_2)(u) = 0$. □

**Lemma 3.4** *For every* infinite *field $\mathcal{K}$ and arbitrary functions $f_1$ and $f_2$*

$$\mathrm{AP}_{\mathcal{K}}\left(f_1 \vee f_2\right) \leq \mathrm{AP}_{\mathcal{K}}(f_1) + \mathrm{AP}_{\mathcal{K}}(f_2) \ .$$

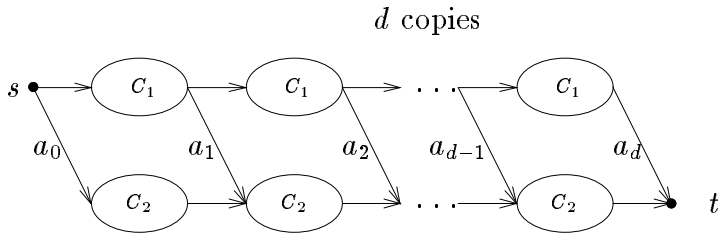


Figure 2: An arithmetic program for $f_1 \vee f_2$, where $\sum_{i=0}^{d} a_i x^i$ is an irreducible polynomial over $\mathcal{K}$. The weights of the diagonal edges are determined by the coefficients of the polynomial.

**Proof:** In the arithmetic program $C$ illustrated in Fig. 3, $C(u) = C_1(u) + \alpha \cdot C_2(u)$. If $(f_1 \vee f_2)(u) = 0$ then $C_1(u) = 0$ and $C_2(u) = 0$ and therefore $C(u) = 0$. If we do not choose $\alpha$ properly then it is possible that $C(u) = 0$ although $(f_1 \vee f_2)(u) = 1$. However, for every assignment $u \in \{0,1\}^n$ such that $(f_1 \vee f_2)(u) = 1$ there exists at most one value of $\alpha$ for which $C_1(u) + \alpha \cdot C_2(u) = 0$. Since $\mathcal{K}$ is infinite, there exists a value for $\alpha$ such that for every $u \in \{0,1\}^n$ it holds that $C_1(u) + \alpha \cdot C_2(u) = 0$ if and only if $C_1(u) = 0$ and $C_2(u) = 0$. □

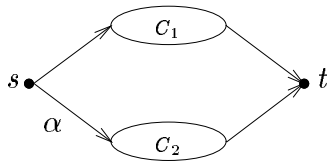


Figure 3: An arithmetic program over an infinite field for $f_1 \vee f_2$, where $\alpha$ is an appropriate constant.

**Corollary 3.5** *Let $k$ be a power of 2. For every field $\mathcal{K}$ and arbitrary functions $f_1, \ldots, f_k$ it holds that* $\mathrm{AP}_{\mathcal{K}}\left(\bigvee_{i=1}^{k} f_i\right) \leq k \sum_{i=1}^{k} \mathrm{AP}_{\mathcal{K}}(f_i)$.

**Proof:** We prove the corollary by induction on $k$. For $k = 2$ there are two cases. If $\mathcal{K}$ is infinite then the corollary for $k = 2$ follows from Lemma 3.4. If $\mathcal{K}$ is finite then there exists an irreducible polynomial of degree 2 over $\mathcal{K}$ (for completeness we present these polynomials in Appendix A). Thus, Lemma 3.3 implies the corollary for $k = 2$ over finite fields.

Assume that the corollary holds for $k/2$. Observe that $\bigvee_{i=1}^{k} f_i = \left(\bigvee_{i=1}^{k/2} f_i\right) \vee \left(\bigvee_{i=k/2+1}^{k} f_i\right)$. Thus by the case $k = 2$ and the induction hypothesis,

$$\mathrm{AP}_{\mathcal{K}}\left(\bigvee_{i=1}^{k} f_i\right) \leq 2\left(\mathrm{AP}_{\mathcal{K}}\left(\bigvee_{i=1}^{k/2} f_i\right) + \mathrm{AP}_{\mathcal{K}}\left(\bigvee_{i=k/2+1}^{k} f_i\right)\right) \leq k \sum_{i=1}^{k} \mathrm{AP}_{\mathcal{K}}(f_i) \ .$$

□

If $k$ is not a power of two then $\mathrm{AP}_{\mathcal{K}}\left(\bigvee_{i=1}^{k} f_i\right) \leq 2k \sum_{i=1}^{k} \mathrm{AP}_{\mathcal{K}}(f_i)$ holds.

# 4 Arithmetic Programs are Equivalent to Complements of Dependency Programs

In [14] it was proved that if a function $f$ has a small arithmetic program then its complement $\overline{f}$ has a small dependency program. In this section we prove the converse; if $\overline{f}$ has a small dependency program then the function $f$ has a small arithmetic program. In particular this implies that dependency programs are closed under conjunction in every field, solving an open problem in [14].

We begin with an arithmetic program that checks if the rank of a given matrix is at least $r$. This arithmetic program is used to construct an arithmetic program which simulates the complement of the dependency program. The construction is based on the results of [6, 12] that show that the computation of the rank of a matrix can be done in $\mathcal{NC}^2$. We start with some notation. We denote by $A(x)$ a matrix in which every entry is either an element of $\mathcal{K}$ or has the form $a \cdot x_i^{\epsilon}$, where $a$ is an element in $\mathcal{K}$ and $x_i^{\epsilon}$ is a literal. Given an assignment $u \in \{0,1\}^n$, the matrix $A(u)$ is a matrix over $\mathcal{K}$ which is obtained from $A(x)$ in the following way:

$$A(u)_{j,k} = \begin{cases} a & \text{if } A(x)_{j,k} = a \\ a & \text{if } A(x)_{j,k} = a \cdot x_i^{\epsilon} \text{ and } u \text{ satisfies } x_i^{\epsilon} \\ 0 & \text{if } A(x)_{j,k} = a \cdot x_i^{\epsilon} \text{ and } u \text{ does not satisfy } x_i^{\epsilon} \ . \end{cases}$$

**Lemma 4.1** *Let $\mathcal{K}$ be any field, and $r, s$ be integers such that $r \leq s$. Furthermore, let $A(x)$ be an $s \times s$ matrix. There exists an arithmetic program of size $s^{O(1)}$ which accepts an input $u$ if and only if* $\mathrm{rank}(A(u)) \geq r$.

**Proof:** The proof is composed of two parts. We first use a result of Mulmuley [12] which reduces the problem of computing the rank to the problem of computing the characteristic polynomial. Then we use an arithmetic program simulating the construction of Berkowitz [6] that computes the coefficients of the characteristic polynomial. There are some technical details left since the result of [12] uses some extension field of $\mathcal{K}$, and we need to simulate the arithmetic over this field by an arithmetic program over $\mathcal{K}$. This simulation uses Corollary 3.5.

We extend the field $\mathcal{K}$ by adding a transcendental element $\alpha$ to obtain $\mathcal{K}(\alpha)$, the field of rational functions in the variable $\alpha$. That is, the elements of the field $\mathcal{K}(\alpha)$ are of the form $p(\alpha)/q(\alpha)$ where $p(\alpha)$ and $q(\alpha)$ are polynomials in $\alpha$ over the field $\mathcal{K}$ (there is no bound on their degrees); addition and multiplication are of polynomials. Following Mulmuley [12], we

transform the $s \times s$ matrix $A(x)$ over $\mathcal{K}$ to an $2s \times 2s$ matrix $B(x)$ over the field $\mathcal{K}(\alpha)$. Let $X$ be a $2s \times 2s$ diagonal matrix where $X_{i,i} = \alpha^{i-1}$ and the other entries are 0. Let

$$B(x) \triangleq X \cdot \begin{pmatrix} O & A(x) \\ (A(x))^T & 0 \end{pmatrix}.$$

Mulmuley [12] proved that for every $u \in \{0,1\}^n$ it holds that $\mathrm{rank}(B(u)) = 2 \cdot \mathrm{rank}(A(u))$; and the rank of the matrix $B(u)$ is $2s - i$, where $i$ is the smallest index such that the coefficient of $\lambda^i$ in the characteristic polynomial of $B$, i.e. $p(\lambda) = \det(\lambda I - B(u))$, is non-zero. Each entry in $B(x)$ has the form $a \cdot \alpha^j$ or $a \cdot x_i^\epsilon \alpha^j$, where $x_i^\epsilon$ is a literal, $a$ is an element in $\mathcal{K}$ and $j < 2s$. Thus, each coefficient of the characteristic polynomial $\det(\lambda I - B(u))$ is a polynomial of $\alpha$ of degree less than $4s^2$.

The result of Berkowitz [6] implies that there is an arithmetic program that checks if a given coefficient of the characteristic polynomial is non-zero (for an alternative construction see Mahajan and Vinay [11]). That is, given a matrix $B(x)$ over the field $\mathcal{K}(\alpha)$ and an integer $\ell$ there is an arithmetic program, denoted $C_\ell$, which accepts all the assignments $u$ such that the coefficient of $\lambda^\ell$ in the characteristic polynomial of $B(u)$ is non-zero. The size of $C_\ell$ is polynomial in $s$ and the weights of the edges in $C_\ell$ are of the form $a \cdot \alpha^j$, corresponding to the entries of $B(x)$, thus $j < 2s$.

We simulate $C_\ell$ with an arithmetic program $C'_\ell$ over $\mathcal{K}$ exploiting the fact that the weights are polynomials of small degree in $\alpha$. Each vertex of $C_\ell$ is duplicated $4s^2$ times, where the $j$-th copy of $v$, denoted by $\langle v, j \rangle$, is "responsible" for the coefficient of $\alpha^j$ (for $0 \leq j \leq 4s^2 - 1$). That is, the sum of the weights of the paths from $\langle s, 0 \rangle$ to $\langle v, j \rangle$ which are consistent with $u$ is equal to the coefficient of $\alpha^j$ in the sum of the weights of the paths from $s$ to $v$ which are consistent with $u$ in $C_\ell$. Formally, for every edge $(z, v)$ with weight $a\alpha^k$ we construct edges $(\langle z, j \rangle, \langle v, j + k \rangle)$ for $0 \leq j \leq 4s^2 - k - 1$. The weight of these edges is $a$ and the label of these edges is the label of $(z, v)$. Thus, we achieve the property that an input $u$ is accepted by $C_\ell$ if and only if there is an index $j$, where $0 \leq j \leq 4s^2 - 1$, such that the sum of the weights of the paths from $\langle s, 0 \rangle$ to $\langle t, j \rangle$ which are consistent with $u$ is non-zero. The arithmetic program $C'_\ell$ over $\mathcal{K}$ will be the disjunction of the subprograms with source $\langle s, 0 \rangle$ and target $\langle t, j \rangle$, respectively, for $0 \leq j \leq 4s^2 - 1$.

To determine if $\mathrm{rank}(A(u)) \geq r$ we check if $\mathrm{rank}(B(u)) \geq 2r$, i.e., if there is a non-zero coefficient among the coefficients $\lambda^0, \lambda^1, \ldots, \lambda^{2s-2r}$ of the characteristic polynomial $\det(\lambda I - B(u))$. Thus, we have to compute the disjunction of the arithmetic programs $C'_\ell$ for $0 \leq \ell \leq 2s - 2r$. We compute the disjunction of arithmetic programs using Corollary 3.5. □

We next describe the arithmetic program for $f$ based on the dependency program for $\overline{f}$.

**Theorem 4.2** *Let $\mathcal{K}$ be any field. Then for an arbitrary function $f$ it holds that* $\mathrm{AP}_{\mathcal{K}}(f) \leq \left(\mathrm{DP}_{\mathcal{K}}\left(\overline{f}\right)\right)^{O(1)}$.

**Proof:** Let $\widehat{M} = \langle M, \rho \rangle$ be a dependency program that computes $\overline{f}$. Suppose that $M$ has $s$ rows. W.l.o.g., we can assume that $M$ is an $s \times s$ matrix (we can either remove columns of $M$ that are linear combinations of other columns of $M$, or duplicate columns). Let $Y(x)$ be an $s \times s$ diagonal matrix, where $Y(x)_{i,i} = \overline{\rho(i)}$ (that is, if $\rho(i) = x_j^{\epsilon}$ then $Y(x)_{i,i} = x_j^{\overline{\epsilon}}$) and the other entries are 0. Furthermore, let $N(x)$ be a matrix where $N(x)_{i,j} = M_{i,j} \cdot \rho(i)$. Now consider the matrix $P(x) \triangleq (N(x), Y(x))$ (that is, $P(x)$ has $s$ rows, $2s$ columns, and the $i$-th row of $P(x)$ is the concatenation of the $i$-th row of $N(x)$ and the $i$-th row of $Y(x)$). If $u$ satisfies $\rho(i)$ (the label of the $i$-th row in $\widehat{M}$) then the $i$-th row in $P(u)$ is $M_i 0^s$. Otherwise, the $i$-th row is $0^{s+i-1}10^{s-i}$. Therefore the rows of $M_u$ are dependent (i.e., the dependency program accepts $u$ and $f(u) = 0$) if and only if the rows of $P(u)$ are dependent. Thus, the arithmetic program, of Lemma 4.1, that accepts an input $u$ if and only if $\mathrm{rank}(P(u)) \geq s$ computes the desired function $f$. □

As we have mentioned already in the introduction, the simulation in the other direction, namely $\mathrm{DP}_{\mathcal{K}}(f) \leq 2 \cdot \mathrm{AP}_{\mathcal{K}}\left(\overline{f}\right)$ follows from the arguments of [14]. The results of [14] are stated only for modular branching programs, however they easily generalize to arithmetic branching programs and imply $\mathrm{DP}_{\mathcal{K}}(f) \leq O\left(\left(\mathrm{AP}_{\mathcal{K}}\left(\overline{f}\right)\right)^2\right)$. An easy modification of the proof in [14] gives $\mathrm{DP}_{\mathcal{K}}(f) \leq 2 \cdot \mathrm{AP}_{\mathcal{K}}\left(\overline{f}\right)$. For completeness we prove this statement in Appendix B.

The next corollary is an immediate consequence of (1), Theorem 4.2 and Corollary 3.5.

**Corollary 4.3** *Let $k$ be a power of 2. For every field $\mathcal{K}$ and arbitrary functions $f_1, \ldots, f_k$ it holds that* $\mathrm{DP}_{\mathcal{K}}\left(\bigwedge_{i=1}^{k} f_i\right) \leq 2k \sum_{i=1}^{k} \left(\mathrm{DP}_{\mathcal{K}}(f_i)\right)^{O(1)}$.

We note that even if the dependency programs for every $f_i$ are monotone (that is, all literals labeling the rows are positive), the resulting dependency program for $\wedge_{i=1}^{k} f_i$ would still be non-monotone. The reason is that the conversion from an arithmetic program to a dependency program results in a non-monotone program. In [14] it is proved that every monotone dependency program for the function $(x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge \ldots \wedge (x_{n-1} \vee x_n)$ has size at least $2^n/n$. Thus, in general we cannot construct a "small" monotone dependency program for $\wedge_{i=1}^{k} f_i$ from small monotone dependency programs for $f_1, \ldots, f_k$.

# 5 Simulating Arithmetic Programs by Span Programs

In this section we show that if a function has a small arithmetic program then it also has a small span program; the size of the resulting span program is only twice the size of the arithmetic program.

**Theorem 5.1** *Let $\mathcal{K}$ be any field, and $f$ be a Boolean function. Then,* $\mathrm{SP}_{\mathcal{K}}(f) \leq 2 \cdot \mathrm{AP}_{\mathcal{K}}(f)$.

**Proof:** Let $\langle G, \mu, w, s, t\rangle$ be an arithmetic program of size $\mathrm{AP}_{\mathcal{K}}(f)$ that computes $f$. We construct a span program that computes $\overline{f}$, and has size not more than twice the size of the arithmetic program. Since $\mathrm{SP}_{\mathcal{K}}(f) = \mathrm{SP}_{\mathcal{K}}(\overline{f})$ this suffices.

Describing the span program requires some more notation. For every vertex $a \in V$ define a vector of length $2^n$ over $\mathcal{K}$, denoted by $\mathbf{f_a}$, in which the $u$-th coordinate in $\mathbf{f_a}$, i.e. $f_a(u)$, is equal to the sum of the weights of the directed paths in $G$ between $s$ and $a$ that are consistent with $u$. Notice that $\overline{f}(u) = 1$ if and only if $f_t(u) = 0$. For a vector $\mathbf{g}$ define the vector $\mathbf{g} \cdot \mathbf{x_i^\epsilon}$ as follows: $(g \cdot x_i^\epsilon)(u) \triangleq g(u)$ if $u_i = \epsilon$ and $(g \cdot x_i^\epsilon)(u) \triangleq 0$ otherwise. For every edge $e = (a, b) \in E$ with label $x_i^\epsilon$ define $\mathbf{f_e} \triangleq \mathbf{f_a} \cdot \mathbf{x_i^\epsilon}$. If the edge is labeled by 1 then $\mathbf{f_e} \triangleq \mathbf{f_a}$. For $B \subseteq E$ – the set of all edges whose head is the vertex $b \in V$ – the following equation holds

$$\mathbf{f_b} = \sum_{e \in B} w(e)\mathbf{f_e} \ , \tag{2}$$

where the arithmetic is over $\mathcal{K}$.

We now construct the following span program $\widehat{M}$ (over $\mathcal{K}$). For every edge $e = (a, b)$ in the arithmetic program which is labeled by a literal $x_i^\epsilon$, we have the following two rows in the span program: The row $\mathbf{f_a} \cdot \mathbf{x_i^0}$ labeled by $x_i^1$ and the row $\mathbf{f_a} \cdot \mathbf{x_i^1}$ labeled by $x_i^0$. (Edges labeled by 1 are ignored.) The target vector is $\mathbf{f_t}$. To prove that $\widehat{M}$ computes $\overline{f}$ we have to prove that $\overline{f}(u) = 1$ if and only if $\mathbf{f_t} \in \mathrm{span}(M_u)$. First assume that $\overline{f}(u) = 0$, i.e., $f_t(u) \neq 0$. Thus, the $u$-th coordinate in $\mathbf{f_t}$ is not zero. By the construction of the span program, in all the rows that are consistent with $u$ the $u$-th coordinate is 0, hence they cannot span the vector $\mathbf{f_t}$.

Next we prove the following claims by induction.

$$\forall u \in \{0,1\}^n \ \ \forall a \in V \qquad \mathbf{f_a} - f_a(u)\mathbf{1} \in \mathrm{span}(M_u) \ . \tag{3}$$

$$\forall u \in \{0,1\}^n \ \ \forall e \in E \qquad \mathbf{f_e} - f_e(u)\mathbf{1} \in \mathrm{span}(M_u) \ . \tag{4}$$

Before proving these claims we show that they imply our statement. Let $\overline{f}(u) = 1$, then by the definition of arithmetic programs $f_t(u) = 0$. Therefore, (3) implies that $\mathbf{f_t} \in \mathrm{span}(M_u)$ as required.

The proof of both claims is by induction on the vertices and the edges according to an appropriate ordering. First we prove that (3) holds for the vertex $s$. By definition $\mathbf{f_s} = \mathbf{1}$, thus for every $u$ it holds that $f_s(u) = 1$ and $\mathbf{f_s} - f_s(u)\mathbf{1} = \mathbf{0} \in \mathrm{span}(M_u)$.

Assume that (3) is true for a vertex $a$ and consider the edge $e = (a, b)$. If the edge is labeled by 1 then $\mathbf{f_e} = \mathbf{f_a}$ and the claim is trivial. Otherwise, let $e = (a, b)$ be an edge labeled by a literal $x_i^\epsilon$. There are two cases.

The first case is when $\epsilon \neq u_i$, thus $f_e(u) = 0$. So, $\mathbf{f_e} - f_e(u)\mathbf{1} = \mathbf{f_e} = \mathbf{f_a} \cdot \mathbf{x_i^\epsilon} = \mathbf{f_a} \cdot \mathbf{x_i^{\overline{u}_i}}$ . The vector $\mathbf{f_a} \cdot \mathbf{x_i^{\overline{u}_i}}$ is a row in the span program labeled by $x_i^{u_i}$. Since $x_i^{u_i}(u) = 1$ this row belongs to $M_u$. Thus, in this case $\mathbf{f_e} - f_e(u)\mathbf{1} \in \mathrm{span}(M_u)$.

The second case is when $\epsilon = u_i$, i.e., $f_e(u) = f_a(u)$. By definition it holds that $\mathbf{f_e} = \mathbf{f_a} \cdot \mathbf{x_i^\epsilon} = \mathbf{f_a} - \mathbf{f_a} \cdot \mathbf{x_i^{\overline{\epsilon}}}$. Thus, $\mathbf{f_e} - f_e(u)\mathbf{1} = \mathbf{f_a} - \mathbf{f_a} \cdot \mathbf{x_i^{\overline{\epsilon}}} - f_a(u)\mathbf{1}$ . By the induction hypothesis $\mathbf{f_a} - f_a(u)\mathbf{1} \in \text{span}(M_u)$, and by the construction of $\widehat{M}$ it holds that $\mathbf{f_a} \cdot \mathbf{x_i^{\overline{\epsilon}}} \in \text{span}(M_u)$, therefore (4) follows for this case.

We now prove the induction step for vertices. Let $b$ be a vertex and $B$ the set of its incoming edges. Assume that every edge in $B$ satisfies (4). Using Equation (2) we deduce

$$\mathbf{f_b} - f_b(u)\mathbf{1} = \sum_{e \in B} w(e)\mathbf{f_e} - \left( \sum_{e \in B} w(e) f_e(u) \right) \mathbf{1} = \sum_{e \in B} w(e) \left( \mathbf{f_e} - f_e(u)\mathbf{1} \right) \ .$$

Hence, $\mathbf{f_b} - f_b(u)\mathbf{1}$ is a linear combination of vectors which according to the induction hypothesis are in $\text{span}(M_u)$. So, $\mathbf{f_b} - f_b(u)\mathbf{1}$ is also in $\text{span}(M_u)$. This completes proving both claims. We have proved that the span program $\widehat{M}$ accepts exactly those inputs that satisfy $\overline{f}$, as desired. □

Theorem 5.1 is a generalization of a result of Karchmer and Wigderson [10], who proved the theorem over GF(2). The construction of the span program in our proof is a generalization of their construction with a somewhat simpler proof. The constant 2 in Theorem 5.1 cannot be replaced by any smaller constant. For a field $\mathcal{K}$ consider the function that accepts $u$ if $\sum_{i=1}^{n} u_i \neq 1$ (where the sum is in $\mathcal{K}$). This function has an arithmetic program of size $n$ over $\mathcal{K}$, however, every span program computing it has size at least $2n$.

# 6 Simulating Nondeterministic Branching Programs by Arithmetic Programs

In this section we prove that nondeterministic branching programs can be efficiently simulated by arithmetic programs over any field. Wigderson [19] proved that modular branching programs over GF(2) (and any fixed finite field) can simulate nondeterministic branching programs (e.g., $\mathcal{NL}/\text{poly} \subseteq \oplus\mathcal{L}/\text{poly}$). The construction of Wigderson [19] directly generalizes to arithmetic programs over arbitrary fields, and yields arithmetic programs of size $O\left(n|V|^2 \left(\text{NBP}(f)\right)^3\right)$, where $n$ is the number of variables, $|V|$ is the number of vertices and $\text{NBP}(f)$ is the size of the smallest the nondeterministic branching program computing $f$. We give an alternative construction which is more efficient, i.e., the resulting arithmetic program is smaller (see Theorem 6.4).

We note that there are different definitions for nondeterministic branching programs in different papers. For more information on nondeterministic branching programs the reader can refer for example to [15]. In this paper a nondeterministic branching program is a quadruple $\langle G, \mu, s, t \rangle$ where $G$ is a directed acyclic graph, $\mu$ is a labeling of the edges by literals and $s$ and $t$ are vertices. The nondeterministic branching program accepts $u$ if there is at least one path in $G$ from $s$ to $t$ which is consistent with $u$. The size of the

nondeterministic branching program is the number of edges in the graph. We denote by $\mathrm{NBP}(f)$ the size of the smallest nondeterministic branching program computing $f$.

A simple idea to simulate nondeterministic branching programs by arithmetic programs is to give the weight 1 to every edge in the nondeterministic branching program computing $f$. The sum of the weights of the paths consistent with an assignment $u$ is equal to the number of the consistent paths. If $f(u) = 0$ then there are no consistent paths in the nondeterministic branching program for $f$ and the arithmetic program rejects $u$. However, if the characteristic of the underlying field is not zero, then it is possible that the arithmetic program rejects an assignment $u$ although $f(u) = 1$. Thus, this construction works only over fields of characteristic zero or at least $2^{\mathrm{NBP}(f)}$.

The idea of our construction is similar. We randomly choose a weight for each edge uniformly and independently from a subset of the field. This defines an arithmetic program. If the subset is big enough, but still not too big, then the probability that the arithmetic program accepts an input such that $f(u) = 1$ is high. Furthermore, the program rejects all inputs such that $f(u) = 0$. We will show that this implies that $f$ can be computed as a disjunction of less than $n$ functions, each one having an arithmetic program of size at most $\mathrm{NBP}(f)$. Unlike for the previous construction, the characteristic of the field can be small. For a "small" field, we first construct an arithmetic program over a large enough extension field, and then simulate it by an arithmetic program over the original field (similarly to the simulation in Lemma 4.1). We now formalize these ideas.

**Lemma 6.1** *Let $G = \langle V, E \rangle$ be a directed acyclic graph, $s$ and $t$ be two vertices in the graph, and $A \subseteq \mathcal{K}$ be a finite set. For a weight function $w : E \to A$ define the weight of a path in $G$ as the product of the weights of the edges in the path, and denote the sum of the weights of all directed paths from $s$ to $t$ in $G$ by $\mathrm{wt}_w(G, s, t)$. If there is a path from $s$ to $t$ in $G$ and we choose $w$ uniformly from the space of functions from $E$ to $A$ then $\Pr[\mathrm{wt}_w(G, s, t) = 0] < |V|/|A|$ .*

**Proof:** We prove, by induction, that for every $a \in V$ if the length of the shortest path from $s$ to $a$ is $\ell$ then $\Pr[\mathrm{wt}_w(G, s, a) = 0] \leq \ell/|A|$ . The basis is trivial since by definition $\mathrm{wt}_w(G, s, s) = 1$. For the induction step, assume that the length of the shortest path from $s$ to $b$ is $\ell + 1$, and let $a$ be the last vertex before $b$ on a shortest path from $s$ to $b$. For a weight function $w$ let $\beta_w$ be the sum of the weights of all paths from $s$ to $b$ that do not pass through the edge $(a, b)$. Notice, that both $\mathrm{wt}_w(G, s, a)$ and $\beta_w$ are independent of $w((a, b))$. Using this notation $\mathrm{wt}_w(G, s, b) = \mathrm{wt}_w(G, s, a) \cdot w((a, b)) + \beta_w$. Thus, if $\mathrm{wt}_w(G, s, a) \neq 0$ then there is exactly one value of $w((a, b))$ such that $\mathrm{wt}_w(G, s, b) = 0$. By the induction hypothesis, the probability that $\mathrm{wt}_w(G, s, a) = 0$ is at most $\ell/|A|$ and the probability that $w((a, b)) = -\beta_w/\mathrm{wt}_w(G, s, a)$ is at most $1/|A|$. □

**Lemma 6.2** *Let $\mathcal{K}$ be a field with at least $T \cdot \mathrm{NBP}(f)$ elements, and $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. Then, $\mathrm{AP}_{\mathcal{K}}(f) = O\left(\lceil n/\log T \rceil^2 \mathrm{NBP}(f)\right)$.*

**Proof:** Let $\langle G, \mu, s, t\rangle$ be a nondeterministic branching program of size $\mathrm{NBP}(f)$ computing $f$. Denote $k \triangleq \left\lceil \frac{n}{\log T} \right\rceil$. For an assignment $u$ let $G(u)$ be the graph that contains all edges of $G$ that have labels satisfied by $u$. Let $A \subseteq \mathcal{K}$ be a set of size $\lceil T \cdot \mathrm{NBP}(f) \rceil$. We choose a weight function $w$ uniformly from the space of functions from $E$ to $A$. This defines an arithmetic program. If $f(u) = 0$ then there is no path from $s$ to $t$ in $G(u)$ (since the nondeterministic branching program computes $f$). Thus, $\mathrm{wt}_w(G(u), s, t) = 0$, and the arithmetic program rejects $u$. On the other hand, if $f(u) = 1$ then there is at least one path from $s$ to $t$ in $G(u)$. By Lemma 6.1, the probability that $\mathrm{wt}_w(G(u), s, t) = 0$ is less than $1/T$.

Take $k$ copies of this arithmetic program, each copy with an independent random weight function, and denote the function computed by the $j$-th copy by $g_j$. The probability that there is an input $u$ such that $f(u) = 1$ but $g_j(u) = 0$ for every $j$ is less than $2^n T^{-k} \leq 1$. Thus, there exists $k$ weight functions such that $f = \bigvee_{j=1}^{k} g_j$, and $\mathrm{AP}_{\mathcal{K}}(f) = O\left(k \sum_{j=1}^{k} \mathrm{AP}_{\mathcal{K}}(g_j)\right) = O\left(k \sum_{j=1}^{k} \mathrm{AP}_{\mathcal{K}}(f)\right) = O\left(k^2 \cdot \mathrm{AP}_{\mathcal{K}}(f)\right)$. □

We next claim that arithmetic programs over $\mathrm{GF}(q^d)$ can be simulated by arithmetic programs over $\mathrm{GF}(q)$. The proof is similar to the simulation in Lemma 4.1. A similar result for span programs was proved in [10] (see also [4]).

**Lemma 6.3** *Let $d$ be a positive integer and $q$ be a prime-power. Then,* $\mathrm{AP}_{\mathrm{GF}(q)}(f) \leq 2d^4 \cdot \mathrm{AP}_{\mathrm{GF}(q^d)}(f)$.

**Proof:** Each element in $\mathrm{GF}(q^d)$ can be represented as a polynomial $p(\alpha)$ over $\mathrm{GF}(q)$ of degree at most $d-1$. The addition of elements in $\mathrm{GF}(q^d)$ is the addition of the polynomials and the multiplication is the multiplication of the polynomials modulo a fixed irreducible polynomial in $\mathrm{GF}(q)$ of degree $d$. The simulating program simulates this arithmetic.

As the first step for constructing the arithmetic program over $\mathrm{GF}(q)$, every vertex of the arithmetic program over $\mathrm{GF}(q^d)$ is duplicated $d$ times. The $j$-th copy of $v$, denoted by $\langle v, j\rangle$, is "responsible" for the coefficient of $\alpha^j$ (for $0 \leq j \leq d-1$). That is, we construct the arithmetic program over $\mathrm{GF}(q)$ such that the sum of the weights of the consistent paths from $\langle s, 0\rangle$ to $\langle v, j\rangle$ is equal to the coefficient of $\alpha^j$ in the sum of the weights of the consistent paths from $s$ to $v$. Let $(z, v)$ be an edge with weight $p(\alpha)$ of the original program. For every $j = 0, \ldots, d-1$ let $\alpha^j \cdot p(\alpha) \triangleq \sum_{\ell=0}^{d-1} p_{j,\ell}\alpha^\ell$ (where $\alpha^j \cdot p(\alpha)$ is reduced modulo the fixed irreducible polynomial). In the arithmetic program over $\mathrm{GF}(q)$ we add the $d^2$ edges $(\langle z, j\rangle, \langle v, k\rangle)$ for $0 \leq j, k \leq i-1$. The label of these $d^2$ edges is the same as the label of the edge $(z, v)$, and the weight of the edge $(\langle z, j\rangle, \langle v, k\rangle)$ is $p_{j,k}$.

The arithmetic program over $\mathrm{GF}(q^d)$ rejects an input $u$ if and only if for every $0 \leq j \leq d-1$ the sum of the weights of the consistent paths from $\langle s, 0\rangle$ to $\langle t, j\rangle$ in the arithmetic program over $\mathrm{GF}(q)$ is zero. Denote by $f_j$ the function computed by the arithmetic program over $\mathrm{GF}(q)$ when the target vertex is $\langle t, j\rangle$. Since we included $d^2$ edges for each original edge we have $\mathrm{AP}_{\mathrm{GF}(q)}(f_j) \leq d^2 \cdot \mathrm{AP}_{\mathrm{GF}(q^d)}(f)$. Furthermore, $f = \bigvee_{j=1}^{d-1} f_j$. Using Corollary 3.5 we get $\mathrm{AP}_{\mathrm{GF}(q)}(f) \leq 2d \cdot \sum_{j=1}^{d-1} \mathrm{AP}_{\mathrm{GF}(q)}(f_j) \leq 2d^4 \cdot \mathrm{AP}_{\mathrm{GF}(q^d)}(f)$. □

**Theorem 6.4** *For any field $\mathcal{K}$ and an arbitrary Boolean function $f$ in $n$ variables*

$$\mathrm{AP}_{\mathcal{K}}(f) \leq O\left(n^2 \cdot \mathrm{NBP}(f) \cdot (\log \mathrm{NBP}(f))^2\right).$$

**Proof:** If $\mathcal{K}$ contains at least $(\mathrm{NBP}(f))^2$ elements then the theorem follows from Lemma 6.2. Otherwise, use a finite extension of $\mathcal{K}$ which contains at least $(\mathrm{NBP}(f))^2$ elements and apply Lemma 6.3 to get an arithmetic program over $\mathcal{K}$. □

For GF(2) we can get a smaller arithmetic program of size $O\left(n \cdot \mathrm{NBP}(f) \cdot (\log \mathrm{NBP}(f))^2\right)$. To construct the smaller program first notice that $\mathrm{AP}_{\mathrm{GF}(2)}(f) = \mathrm{AP}_{\mathrm{GF}(2)}\left(\overline{f}\right)$. Therefore, the disjunction of $k$ functions can be computed by a smaller arithmetic program: $\mathrm{AP}_{\mathrm{GF}(2)}\left(\bigvee_{i=1}^{k} f_i\right) = O\left(\sum_{i=1}^{k} \mathrm{AP}_{\mathrm{GF}(2)}(f_i)\right)$ (use De-Morgan laws and the construction for conjunction).

If we are interested in simulating a nondeterministic branching program by a span program we can be more efficient. That is, $\mathrm{SP}_{\mathcal{K}}(f) = O\left(n \cdot \mathrm{NBP}(f)\right)$. We simulate the arithmetic program of Lemma 6.2 over an extension field with at least $2^{2n}$ elements by a span program using Theorem 5.1. The span program over smaller fields is obtained by the result of [10] stating that $\mathrm{SP}_{\mathrm{GF}(q)}(f) \leq d \cdot \mathrm{SP}_{\mathrm{GF}(q^d)}(f)$ .

# 7 Open Problems

We mention three open problems that arise from our work.

**Open Problem 1** *Are arithmetic programs closed under negation over every field? That is, is there a constant $c$ such that $\mathrm{AP}_{\mathcal{K}}\left(\overline{f}\right) = O(\mathrm{AP}_{\mathcal{K}}(f)^c)$ for every field $\mathcal{K}$? In particular, are arithmetic programs closed under negation over the reals or rationals?*

Notice that for every finite field $\mathrm{AP}_{\mathrm{GF}(q)}\left(\overline{f}\right) = O(\mathrm{AP}_{\mathrm{GF}(q)}(f)^{q-1})$. However, we want the constant $c$ to be independent of the size of the field.

**Open Problem 2** *Do span programs and arithmetic programs have the same power over every field? That is, does there exists a constant $c$ such that $\mathrm{AP}_{\mathcal{K}}(f) = O(\mathrm{SP}_{\mathcal{K}}(f)^c)$ for every field $\mathcal{K}$?*

**Open Problem 3** *Is there a constant $c$ such that for every field $\mathcal{K}$, every labeled matrix $A(x)$ with $s$ rows, and every integer $r$ there exists an arithmetic program of size $O(s^c)$ that accepts an input $u \in \{0,1\}^n$ if and only if $\mathrm{rank}(A(u)) = r$?*

If arithmetic programs are closed under complement then there is an arithmetic program that checks if the rank of a matrix is exactly $r$ (and not only at least $r$). This would enable to simulate span programs by arithmetic programs. On the other hand, if arithmetic programs

are equivalent to span programs then they are closed under complement since span programs are closed under complement. Thus, the three open problems are equivalent. We note that the uniform version of these problems over the rationals is equivalent to an open problem in [2].

We should mention that span programs and arithmetic programs are equivalent over fixed finite fields up to polynomial factors. The size of the arithmetic program simulating the span program depends on the number of elements in the field.

**Acknowledgments:** We would like to thank Eric Allender, Eyal Kushilevitz, and Avi Wigderson for helpful discussions.

# References

[1] E. Allender. Making computation count: Arithmetic circuits in the nineties. *SIGACT NEWS, Complexity Theory Column*, 1997. To appear.

[2] E. Allender, R. Beals, and M. Ogihara. The complexity of matrix rank and feasible systems of linear equations. In *Proc. of the 28th Annu. ACM Symp. on the Theory of Computing*, pages 161–167, 1996.

[3] L. Babai, A. Gál, J. Kollár, L. Rónyai, T. Szabó, and A. Wigderson. Extremal bipartite graphs and superpolynomial lower bounds for monotone span programs. In *Proc. of the 28th Annu. ACM Symp. on the Theory of Computing*, pages 603–611, 1996.

[4] L. Babai, A. Gál, and A. Wigderson. Superpolynomial lower bounds for monotone span programs. Submitted for publication, 1996.

[5] A. Beimel, A. Gál, and M. Paterson. Lower bounds for monotone span programs. *Computational Complexity*, 6(1):29–45, 1997. Conference version: FOCS '95.

[6] S. J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Inform. Process. Lett.*, 18:147–150, 1984.

[7] G. Buntrock, C. Damm, H. Hertrampf, and C. Meinel. Structure and importance of the logspace-mod class. Math. Systems Theory, 25:223–237, 1992.

[8] J. F. Buss, G. S. Frandsen, and J. O. Shallit. The computational complexity of some problems of linear algebra. In R. Reischuk and M. Morvan, editors, *STACS '97*, volume 1200 of *Lecture Notes in Computer Science*, pages 451–462. Springer, 1997. See also: ECCC report TR97-009, http://www.eccc.uni-trier.de/eccc/.

[9] Y. Ishai and E. Kushilevitz. Private simultaneous messages protocols with applications. In *5th Israel Symp. on Theory of Computing and Systems*, pages 174–183, 1997.

[10] M. Karchmer and A. Wigderson. On span programs. In *Proc. of the 8th Annu. IEEE Structure in Complexity Theory*, pages 102–111, 1993.

[11] M. Mahajan and V Vinay. A combinatorial algorithm for the determinant. In *Proc. of the 8th Annu. ACM-SIAM Symp. on Discrete Algorithms*, pages 730–738, 1997.

[12] K. Mulmuley. A fast parallel algorithm to compute the rank of a matrix over an arbitrary field. *Combinatorica*, 7:101–104, 1987.

[13] N. Nisan. Lower bounds for non-commutative computation. In *Proc. of the 23th Annu. ACM Symp. on the Theory of Computing*, pages 410–418, 1991.

[14] P. Pudlák and J. Sgall. Algebraic models of computation and interpolation for algebraic proof systems. In S. Buss, editor, *Feasible Arithmetic and Lengths of Proofs*, Lecture Notes in Computer Science. Springer, 1996. To appear.

[15] A. A. Razborov. Lower bounds for deterministic and nondeterministic branching programs. In L. Budach, editor, *Proceedings of Fundamentals of Computation Theory (FCT '91)*, volume 529 of *Lecture Notes in Computer Science*, pages 47–60. Springer-Verlag, 1991.

[16] K. Reinhardt and E. Allender. Making nondeterminism unambiguous. In *Proc. of the 38th Annu. IEEE Symp. on Foundations of Computer Science*, pages 244–253, 1977.

[17] C. Small. *Arithmetic of finite fields*. M. Dekker, 1991.

[18] V. Strassen. Algebraic complexity theory. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 11, pages 635–672. Elsevier and The MIT press, 1990.

[19] A. Wigderson. $NL/poly \subseteq \oplus L/poly$. In *Proc. of the 9th Annu. IEEE Structure in Complexity Theory*, pages 59–62, 1994. Journal version: A. Gál and A. Wigderson, Boolean Complexity Classes vs. Their Arithmetic Analogs, *Random Structures and Algorithms*, 9:99–111, 1996.

# A Irreducible Polynomials in Finite Fields

In this section we show that every finite field has an irreducible polynomial of degree 2. This in an elementary result in the theory of finite fields (see e.g., [17]), and we include its proof

for completeness. There are two cases to consider: fields of odd characteristic and fields of characteristic two. We start with the fields of odd characteristic. A number $a$ is a quadratic non-residue in a field $\mathcal{K}$ if it is not a square of any number in $\mathcal{K}$ (i.e. $b^2 \neq a$ for every $b \in \mathcal{K}$). In every finite field GF($q$), where $q$ is a power of an odd prime, there are $(q-1)/2$ quadratic non-residues (since every quadratic-residue is a root of the polynomial $x^{(q+1)/2} - x$). The following is an immediate consequence of the definition of quadratic non-residues.

**Lemma A.1** *Let $q$ be a power of an odd prime, and $a$ be a quadratic non-residue in* GF($q$). *The polynomial $x^2 - a$ is irreducible.*

The arithmetic program for $f_1 \vee f_2$, as proved in Lemma 3.3, is described in Fig. 4. There

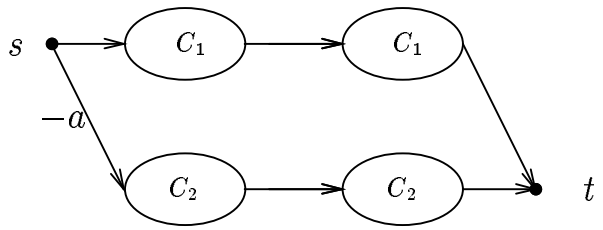


Figure 4: An arithmetic program for $f_1 \vee f_2$, where $a$ is a quadratic non-residue.

are no quadratic non-residues in finite fields of characteristic two. In these fields the irreducible polynomial of degree 2 is different. We use the following notation. The trace of an element $a \in \text{GF}(2^i)$, denoted by $T(a)$, is defined as

$$T(a) \triangleq \sum_{j=0}^{i-1} a^{2^j} \ .$$

**Lemma A.2** *Let $i$ be a positive integer. There exists an element $a \in$ GF($2^i$) such that $T(a) \neq 0$. Furthermore, for every $a$ such that $T(a) \neq 0$ the polynomial $x^2 + x + a$ is irreducible in* GF($2^i$).

**Proof:** Every element $x$ such that $T(x) = 0$ is a root of the polynomial $\sum_{j=0}^{i-1} x^{2^j}$, which has at most $2^{i-1}$ roots. Thus, there are at least $2^{i-1}$ elements in GF($q$) such that $T(a) \neq 0$. (For example, if $i$ is odd then $T(1) = 1 \neq 0$.)

Consider the polynomial $x^2 + x + a$. If this polynomial is reducible then it has a root $b$, that is, $a = b^2 + b$. Thus,

$$T(a) \ = \ \sum_{j=0}^{i-1} a^{2^j} = \sum_{j=0}^{i-1} (b^2 + b)^{2^j}$$

$$= \sum_{j=0}^{i-1} \left( (b^2)^{2^j} + b^{2^j} \right) = \sum_{j=0}^{i-1} b^{2^{j+1}} + \sum_{j=0}^{i-1} b^{2^j}$$
$$= b^{2^i} + b = b + b = 0 \ .$$

But $a$ was chosen such that $T(a) \neq 0$, and $x^2 + x + a$ is irreducible. □

The arithmetic program for $f_1 \vee f_2$ over GF($2^i$), as proved in Lemma 3.3, is described in Fig. 5.

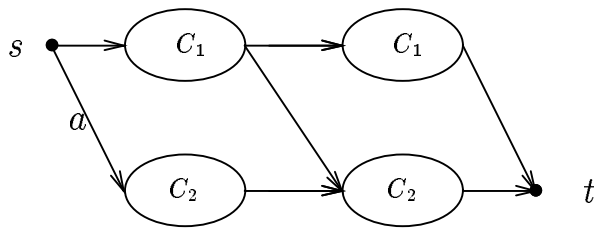


Figure 5: An arithmetic program over a finite field of characteristic two for $f_1 \vee f_2$, where $T(a) \neq 0$.

# B Simulating Complements of Arithmetic Programs by Dependency Programs

In this section we show how to construct from an arithmetic program computing $f$ a dependency program computing $\overline{f}$, with only doubling the size. The proof is based on the arguments of [14].

**Theorem B.1** *Let $\mathcal{K}$ be any field. For an arbitrary function $f$*

$$\mathrm{DP}_{\mathcal{K}}\left(\overline{f}\right) \leq 2 \cdot \mathrm{AP}_{\mathcal{K}}(f) \ .$$

**Proof:** Given an arithmetic program $C = \langle G, \mu, w, s, t \rangle$ of size $\mathrm{AP}_{\mathcal{K}}(f)$ computing $f$ we construct a dependency program of size at most $2 \cdot \mathrm{AP}_{\mathcal{K}}(f)$ computing $\overline{f}$. We first show how to construct a matrix $B(x)$ such that the rows of the matrix $B(u)$ are linearly dependent if and only the arithmetic program rejects the input $u$. This part of the argument is a straightforward generalization of the arguments of [14] to arithmetic programs. Next we transform this matrix into a dependency program, similarly to the proof in [14], but reducing the size of the simulation using a simple trick.

The matrix $B(x)$ is obtained from the adjacency matrix of the graph G and the entries of $B(x)$ are of the form $\alpha \cdot x_i^{\epsilon}$. We first describe an intermediate matrix $A(x)$ whose rows

and columns are indexed by vertices of $G$ ordered topologically from $s$ to $t$. For two vertices $a$ and $b$ in $G$ the $a, b$ entry of $A(x)$ is defined as $w((a,b)) \cdot \mu((a,b))$. We obtain the matrix $B(x)$ from $A(x) - I$ by deleting the first column (indexed by $s$) and the last column (indexed by $t$). In [7] it is proved that

**Lemma B.2** [7] *For every $u \in \{0,1\}^n$ it holds that $C(u) = \pm \det(B(u))$ (where $C(u)$ is the sum of the weights of the paths that are consistent with $u$ in the arithmetic program $C$).*[2]

In [7] the lemma is stated for mod-$p$ branching programs, but the proof applies to arithmetic programs with arbitrary weights.

The matrix $B(x)$ is "nearly" our desired dependency program for $\overline{f}$; the rows of $B(u)$ are dependent (that is, $\det(B(u)) = 0$) if and only if $\overline{f}(u) = 1$. However, $B(x)$ may contain several literals in each row. To overcome this problem we first modify the arithmetic program that we started with. We add extra vertices and edges (labeled by the constant 1) to the graph $G$ such that for every vertex either there is one literal that labels all its outgoing edges, or all these edges are labeled by 1. The number of edges labeled by literals is not changed during this process. After this transformation for every row of $B(x)$ there is a literal such that each entry of the row is either a constant from $\mathcal{K}$ or the product of a constant and the literal.

We are ready to define the dependency program $\widehat{M}$. We simply need to ensure that $M_u = B(u)$. Then the rows of $M_u$ are dependent if and only if $\overline{f}(u) = 1$, i.e., the dependency program computes $\overline{f}$ as required. The construction of $\widehat{M}$ generates (at most) two rows from every row of $B(x)$. If a row of $B(x)$ contains no literals then this row is copied to $\widehat{M}$ and is labeled by 1. For each row of $B(x)$ with label $x_i^\epsilon$ we construct two rows in the dependency program. The first row, labeled by $x_i^{\overline{\epsilon}}$, keeps the entries of the original row that are constants (field elements) and sets the other entries to zero. The second row, labeled by $x_i^\epsilon$, keeps the constant part of each entry of the original row. (For an entry of the form $\alpha \cdot x_i^\epsilon$ the corresponding coordinate of the second row is $\alpha$.) Finally, we can eliminate the rows labeled by 1 without changing the number of rows labeled by a literal (for details see [14]). We finished describing the dependency program computing $\overline{f}$, which completes the proof of the theorem. □

[2] It can be proved that actually $C(u) = \det(B(u))$.