On Arithmetic Branching Programs*

Amos Beimel†

Division of Engineering & Applied Sciences
Harvard University
40 Oxford st., Cambridge, MA 02138.

Anna Gál‡

Dept. of Computer Sciences
The University of Texas at Austin
Austin, TX 78712.

Abstract

The model of arithmetic branching programs is an algebraic model of computation generalizing the model 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 [20]. Using this equivalence we prove that dependency programs are closed under conjunction over every field, answering an open problem of [20]. Furthermore, we show that span programs, an algebraic model of computation introduced by Karchmer and Wigderson [16], 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 [25]. Our simulation of $\mathcal{NL}/\text{poly}$ is more efficient, and it holds for logspace counting classes over every field.

Key Words: Algebraic Models of Computation, Arithmetic Branching Programs, Span Programs, Dependency Programs, Nondeterministic Branching Programs.

1 Introduction

Algebraic models of computation have received much attention (e.g., [24, 13, 14, 1]). One fundamental question is the power of basic algebraic operations. For example, if we can

*Preliminary version of this paper appeared in the proceedings of the 13th Annu. IEEE conference on Computational Complexity, pages 68–80, 1998.

†Email: beimel@deas.harvard.edu. Supported by grants ONR-N00014-96-1-0550 and ARO-DAAL03-92-G0115. Part of this work was done while the author was a postdoctoral fellow at DIMACS, supported in part by NSF under contract STC-91-19999 and the New Jersey Commission on Science and Technology.

‡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, supported in part by NSF under contract STC-91-19999 and the New Jersey Commission on Science and Technology, and while on leave at the Dept. of Computer Science and the Fields Institute at the University of Toronto, partially supported by ITRC, an Ontario Centre of Excellence.

efficiently compute the rank of a matrix then we can efficiently check if the rows of the matrix are dependent. Can we efficiently 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 models of computation. In this work we study algebraic models which supply an arena to study such questions.

The algebraic models we consider are arithmetic branching programs, span programs [16] and dependency programs [20]. (We give precise definitions of the above models in the next section.) Karchmer and Wigderson [16] defined span programs and showed, using results of [9], that for every fixed prime p span programs over the finite fields $\mathrm{GF}(p)$ have the same power as modular branching programs. That is, a function can be computed by a polynomial size mod-p branching program if and only if it can be computed by a polynomial size span program over $\mathrm{GF}(p)$. Span programs were considered also in [5, 3, 4, 2, 12, 20]. Pudlák and Sgall [20] defined a similar model called dependency programs. They proved that span programs are at least as strong as dependency programs over every field, and for fixed finite fields the reverse also holds, that is over fixed finite fields span programs and dependency programs are equivalent. ([20] considered only fields $\mathrm{GF}(p)$ for p prime, but their arguments easily generalize to arbitrary fixed finite fields.) It is not known whether span programs are stronger than dependency programs over "large" fields, in particular over the reals or rationals.

The equivalence between modular branching programs, span programs and dependency programs over $\mathrm{GF}(p)$, for a fixed prime p, is very useful in understanding these models. While span programs and dependency programs are defined over arbitrary fields, modular branching programs can be defined only over the fields $\mathrm{GF}(p)$ for a prime p. In this work, we focus on the model of arithmetic branching programs which is a generalization of modular branching programs and is defined over arbitrary fields. The model of arithmetic branching programs sheds more light on the models of span programs and dependency programs, e.g., we prove that dependency programs are closed under conjunction, over arbitrary fields, using arithmetic programs.[1] Furthermore, considering arithmetic branching programs allows us to study the relations between the three models over arbitrary fields.

For clarity of 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; the class of functions that can be computed by polynomial size mod-p branching programs is equal to the class $\mathrm{Mod}_p\mathcal{L}/\mathrm{poly}$ [9, 16]. Modular branching programs have been defined and studied in [9, 16], and were used in [15] to construct private simultaneous protocols for computing functions in $\mathcal{NL}$ and $\mathrm{Mod}_p\mathcal{L}$. Modular branching programs are also referred to in the literature as counting branching programs.

[1] In [20] the closure of dependency programs under conjunction is proved over fixed finite fields; the size of the dependency program computing the conjunction is polynomial in the number of elements in the field and in the sizes of the original dependency programs.

An arithmetic branching program over a field $\mathcal{K}$ (abbreviated arithmetic program) 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.

To summarize the connections between the different models, denote the size of the smallest span program, dependency program and arithmetic program for a function f by $\mathrm{SP}_{\mathcal{K}}(f)$, $\mathrm{DP}_{\mathcal{K}}(f)$, and $\mathrm{AP}_{\mathcal{K}}(f)$ respectively. Pudlák and Sgall [20] 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 [20], however their proof holds for arithmetic programs over arbitrary fields). An easy modification of the arguments in [20] gives

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

(see Appendix B).

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 over any field $\mathcal{K}$

$$\mathrm{AP}_{\mathcal{K}}(f) \leq \left(\mathrm{DP}_{\mathcal{K}}\left(\overline{f}\right)\right)^{O(1)}.$$

This result, together with results of [20], 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 parallel computation of the rank [7, 6, 18] (for a survey on this subject and a complete list of bibliography see [14]) and the closure of arithmetic programs under disjunction. Our results also imply that dependency programs are closed under conjunction over every field, solving an open problem in [20].

Furthermore, we show that span programs can simulate arithmetic programs and dependency programs very efficiently. We prove that 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 [16] for programs over $\mathrm{GF}(2)$. For other fields only a polynomial relation was known [9, 20]. We next prove that over "large" fields span programs can simulate dependency programs without any increase in size. That is, if $\mathcal{K}$ is "large enough" then $\mathrm{SP}_{\mathcal{K}}(f) \leq \mathrm{DP}_{\mathcal{K}}(f)$. We also show that even for small fields the overhead of the simulation can be limited to a multiplicative factor of n, that is $\mathrm{SP}_{\mathcal{K}}(f) \leq n \cdot \mathrm{DP}_{\mathcal{K}}(f)$ over every field. Note that Pudlák and Sgall [20] proved that $\mathrm{SP}_{\mathcal{K}}(f) \leq (\mathrm{DP}_{\mathcal{K}}(f))^2$ and their proof gives a uniform construction, while our construction is non-uniform.

Our last result shows that, over any field, arithmetic programs can efficiently simulate nondeterministic branching programs. This generalizes a result of Wigderson [25] 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 [25] directly generalizes to arithmetic programs over arbitrary fields. The simulation of nondeterministic branching programs by arithmetic programs over any field also follows from a recent result of Reinhardt and Allender [22] who proved, using the construction of [25], that $\mathcal{NL}/\text{poly} = \mathcal{UL}/\text{poly}$. However, we give an alternative construction which is more efficient, i.e., the resulting arithmetic program is smaller.

The models considered in this paper are closely related to complexity classes defined by logspace nondeterministic Turing machines. For a fixed prime p the three models (arithmetic programs, dependency programs and span programs) over GF(p) are equivalent in power to mod-p branching programs, and the class of functions computable by polynomial size programs in these models over GF(p) is equal to the class $\text{Mod}_p\mathcal{L}/\text{poly}$. It is proved in [2] that logspace uniform polynomial size span programs over the rationals characterize the class $\mathcal{L}^{\mathcal{C}_=\mathcal{L}}$. Similarly, logspace uniform polynomial size dependency programs over the rationals characterize the class $\mathcal{C}_=\mathcal{L}$. Our results imply that logspace uniform polynomial size arithmetic programs over the rationals characterize the class co-$\mathcal{C}_=\mathcal{L}$. (See, e.g., [2] for definitions of these complexity classes.)

Remark: Nisan [19] considered a similar model of computation 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 [19] 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, in Section 6 we show that span programs can simulate dependency programs over "large" fields without any increase in size, and in Section 7 we show that over any field arithmetic programs can simulate nondeterministic branching programs with polynomial increase in size. Finally, in Section 8 we summarize our results and discuss some open problems. We present some background results in the appendix.

2 Definitions

In this section we define the three models of computation discussed in this paper: arithmetic branching programs, span programs, and dependency programs. We start with some helpful notation. 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^ϵ 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^ϵ 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 Program) *An* arithmetic branching program *(abbreviated* 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, $\mu : E \rightarrow \{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\} \cup \{1\}$ is a function called the labeling function, $w : E \rightarrow \mathcal{K}$ 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 each path in the graph we assign a weight which is the product *of the weights of the edges in the path. For completeness, we define the weight of the empty path as 1. 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 $\mathrm{AP}_{\mathcal{K}}(f)$.*

Example 2.2 In Fig. 1, we describe an example of an arithmetic program over the rationals. The size of the program is five. For the input $u = 110$ there is exactly one consistent path $s \rightarrow a \rightarrow b \rightarrow t$ whose weight is $3 \cdot 2 \cdot 4 = 24$ and the input is accepted. For the input $u = 100$ there are two consistent paths: $s \rightarrow a \rightarrow b \rightarrow t$ whose weight is 24 and $s \rightarrow b \rightarrow t$ whose weight is -24, thus, the input is rejected.

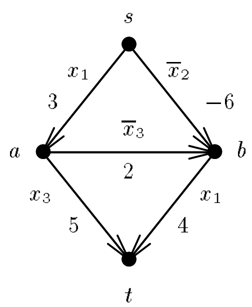

Figure 1: An example of an arithmetic program.

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., [21, 16]).

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 twice the square of the number of edges labeled by a literal [8] (since the number of vertices is at most twice the number of edges labeled by a literal). 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 $\overline{x}_1$. Thus, eliminating edges labeled by 1 would only affect the size of the arithmetic program by a quadratic factor.

A mod-p branching program [9, 16] is an arithmetic program over GF(p) where the weight of every edge is 1. In this case the accepting criterion can be rephrased; the program accepts an assignment if the number of consistent paths is not equal to 0 mod-p.[2]

Definition 2.3 (Span Program [16]) *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 ρ 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 $\mathbf{v} \in \mathrm{span}(M_u)$, 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 $\mathrm{SP}_{\mathcal{K}}(f)$.*

Remark 2.4 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 maximal set of linearly independent columns without changing the function that is computed). Furthermore, for every literal we can assume that the rows labeled by this literal are linearly independent, and their number is at most the number of columns in the program. Thus, without loss of generality, we can assume that the number of rows in the program is at most $2n$ times the number of columns in the program.

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 affect 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 linear space spanned by the rows of the matrix M. The default value for the target vector is $\mathbf{1}$.

Definition 2.5 (Dependency Program [20]) *A* dependency program *over $\mathcal{K}$ is a pair $\widehat{M} = \langle M, \rho\rangle$, where M is a matrix over $\mathcal{K}$, and ρ 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*

[2] In [16] the accepting criterion is different; the program accepts if the number of consistent paths is equal to 1 mod-p. This affects the size of the program by at most a multiplicative factor of $p - 1$.

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* $\mathrm{DP}_{\mathcal{K}}(f)$.

In [20], 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 $2n$ difference in size (see Remark 2.4), 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 can be computed by small arithmetic programs then $f_1 \wedge f_2$ and $f_1 \vee f_2$ can also be computed by 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. 2. $\square$

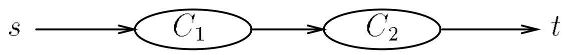

Figure 2: An arithmetic program for $f_1 \wedge f_2$.

The arithmetic program for $f_1 \vee f_2$ is more complicated. The first construction works only over fields which are not algebraically closed. The construction uses the following lemma of [10]. We include its proof for completeness.

Lemma 3.2 ([10]) *Let $d \geq 2$. Assume that $P(x) \triangleq \sum_{i=0}^{d} a_i x^i$ is a polynomial over $\mathcal{K}$ of degree exactly d (i.e., $a_d \neq 0$) such that $P(x)$ has no roots in $\mathcal{K}$, 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 $Q(x_0, y_0) = y_0^d \cdot P(x_0/y_0)$. Thus, $Q(x_0, y_0) = 0$ for $y_0 \neq 0$ implies $P(x_0/y_0) = 0$, contradicting the assumption that $P(x)$ has no roots in $\mathcal{K}$. $\square$

A polynomial is *irreducible* over $\mathcal{K}$ if it is not constant (i.e., it has degree at least 1) and it cannot be written as the product of two non-constant polynomials. If a polynomial of degree at least 2 is irreducible over $\mathcal{K}$ then it has no roots in $\mathcal{K}$. As an example for Lemma 3.2, 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 $d \geq 2$ over $\mathcal{K}$. Then for arbitrary functions f_1 and f_2 it holds that* $\mathrm{AP}_{\mathcal{K}}(f_1 \vee f_2) \leq d\,(\mathrm{AP}_{\mathcal{K}}(f_1) + \mathrm{AP}_{\mathcal{K}}(f_2))$.

Proof: Let C_1 and C_2 be arithmetic programs computing the functions f_1 and f_2 respectively. In Fig. 3, 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 $(f_1 \vee f_2)(u) = 0$. $\square$

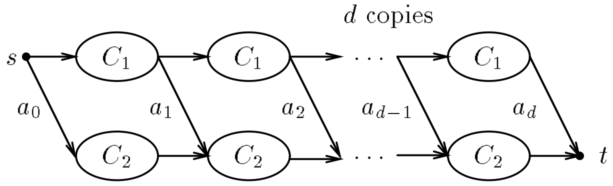

Figure 3: An arithmetic program for $f_1 \vee f_2$, where $\sum_{i=0}^{d} a_i x^i$ is an irreducible polynomial over $\mathcal{K}$. The weights of the diagonal edges are determined by the coefficients of the polynomial.

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

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

Proof: In the arithmetic program C illustrated in Fig. 4, $C(u) = C_1(u) + \alpha \cdot C_2(u)$. 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 α 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 α for which $C_1(u) + \alpha \cdot C_2(u) = 0$. Since $\mathcal{K}$ is infinite, there exists a value for α 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 $C_2(u) = 0$. $\square$

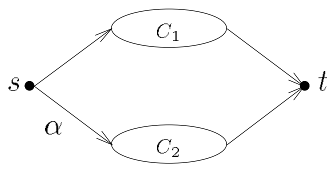

Figure 4: An arithmetic program over an infinite field for $f_1 \vee f_2$, where α is an appropriate constant.

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 three cases. If $\mathcal{K} = \mathrm{GF}(2)$ then the corollary for $k = 2$ follows from Lemma 3.1 and the fact that

$$\mathrm{AP}_{\mathrm{GF}(2)}(f) = \mathrm{AP}_{\mathrm{GF}(2)}\left(\overline{f}\right).$$

If $\mathcal{K}$ is infinite then the corollary for $k = 2$ follows from Lemma 3.4. If $\mathcal{K}$ is finite and $\mathcal{K} \neq \mathrm{GF}(2)$ 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. For comparison, recall that, by Lemma 3.1, $\mathrm{AP}_{\mathcal{K}}\left(\bigwedge_{i=1}^{k} f_i\right) \leq \sum_{i=1}^{k}\mathrm{AP}_{\mathcal{K}}(f_i)$.

Remark 3.6 The construction, given in Lemma 3.3, for the disjunction of arithmetic programs is non-uniform. On the other hand, the construction given in Lemma 3.3 is uniform if an irreducible polynomial of degree $d \geq 2$ over $\mathcal{K}$ can be found efficiently (for example the polynomial $x^2 + 1$ over the rationals and reals). Finding a uniform construction for the disjunction of arithmetic programs over algebraically closed fields and certain finite fields remains open.

Remark 3.7 Over any infinite field $\mathcal{K}$ we have $\mathrm{AP}_{\mathcal{K}}\left(\bigvee_{i=1}^{k} f_i\right) \leq \sum_{i=1}^{k}\mathrm{AP}_{\mathcal{K}}(f_i)$, since we use Lemma 3.4. Furthermore, over the reals and rationals there is a uniform construction for the disjunction of arithmetic programs computing $\bigvee_{i=1}^{k} f_i$ in size $2\sum_{i=1}^{k}\mathrm{AP}_{\mathcal{K}}(f_i)$. For this we use the multivariate polynomial $\sum_{i=1}^{k} y_i^2$, which equals zero if and only if $y_i = 0$ for every i, $1 \leq i \leq k$.

4 Arithmetic Programs are Equivalent to Complements of Dependency Programs

In [20] it was proved that if a function f has a small arithmetic program then its complement $\overline{f}$ has a small dependency program (see Appendix B). In this section we prove the converse; if $\overline{f}$ has a small dependency program then the function f has a small arithmetic program. This implies that dependency programs are closed under conjunction over every field, solving an open problem in [20]. (In [20] the closure under conjunction is proved over fixed finite fields; the size of the dependency program for the conjunction is polynomial in the number of elements in the field and in the sizes of the original dependency programs.)

We present 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 [7, 6, 18] that show that the computation of the rank of a matrix can be done in $\mathcal{NC}^2$ over arbitrary fields. An intermediate stage in the construction uses arithmetic programs over an extension field of $\mathcal{K}$. We show how to simulate these programs by arithmetic programs over $\mathcal{K}$.

Notation: Let $A(x)$ denote 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 of $\mathcal{K}$ and x_i^ϵ 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 $A(x)$ by substituting the values u_i for x_i, that is:

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

Let $\mathcal{K}$ be a field. $\mathcal{K}[\alpha]$ denotes the ring of polynomials in α with coefficients from $\mathcal{K}$. For a polynomial $p(\alpha) \in \mathcal{K}[\alpha]$ of degree d, $\mathcal{K}[\alpha]/p(\alpha)$ denotes the ring of polynomials in α of degree at most $d-1$, where addition and multiplication are of polynomials modulo the polynomial $p(\alpha)$. Furthermore, $\mathcal{K}(\alpha)$ denotes the field of rational functions in the variable α. That is, the elements of the field $\mathcal{K}(\alpha)$ are of the form $p(\alpha)/q(\alpha)$ where $p(\alpha), q(\alpha) \in \mathcal{K}[\alpha]$ (there is no bound on their degrees); addition and multiplication are of polynomials.

4.1 Simulating Arithmetic Programs over Smaller Fields

In this section we show how to simulate arithmetic programs over certain extension fields of a field $\mathcal{K}$ by arithmetic programs over the field $\mathcal{K}$. In the process, it will be convenient to consider arithmetic programs over rings instead of fields. The definition of arithmetic programs (Definition 2.1) remains meaningful over rings since the definition only requires addition and multiplication. (This is similar to the model of algebraic branching programs introduced by Nisan [19] which was defined over rings.)

Lemma 4.1 *Let $\mathcal{K}$ be any field and $p(\alpha) \in \mathcal{K}[\alpha]$ be a polynomial of degree d. Then,*

$$\mathrm{AP}_{\mathcal{K}}(f) \leq 2d^4 \cdot \mathrm{AP}_{\mathcal{K}[\alpha]/p(\alpha)}(f).$$

Proof: Let C be a program over $\mathcal{K}[\alpha]/p(\alpha)$ that computes f. We simulate C with an arithmetic program over $\mathcal{K}$ exploiting the fact that the weights are polynomials in α of degree at most $d-1$. As an intermediate stage we construct d arithmetic programs $C_0, C_1, \ldots, C_{d-1}$ over $\mathcal{K}$. These programs differ only in the target vertex. To obtain C_i, each vertex of C is duplicated d times, where the j-th copy of v is denoted by $\langle v, j\rangle$. Intuitively, the vertex $\langle v, j\rangle$ is "responsible" for the coefficient of α^j (for $0 \leq j \leq d-1$). The source vertex is $\langle s, 0\rangle$ and the target vertex in C_i is $\langle t, i\rangle$. Let (z, v) be an edge in C whose weight is $q(\alpha) \in \mathcal{K}[\alpha]/p(\alpha)$. For every $j = 0, \ldots, d-1$ let

$$\alpha^j \cdot q(\alpha) \triangleq \sum_{k=0}^{d-1} q_{j,k}\alpha^k \tag{1}$$

(where $\alpha^j \cdot q(\alpha)$ is reduced modulo the polynomial $p(\alpha)$). In the arithmetic program C_i we add the d^2 edges $(\langle z,j\rangle, \langle v,k\rangle)$ for $0 \le j,k \le d-1$. 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 $q_{j,k}$.

By simple induction, for every input u the sum of the weights of the consistent paths from $\langle s,0\rangle$ to $\langle v,j\rangle$ in C_i is equal to the coefficient of α^j in the sum of the weights of the consistent paths from s to v in C. The arithmetic program C rejects an input u if and only if the sum of the weights of the consistent paths from s to t in C is equal to the zero polynomial, that is, if and only if for every $0 \le i \le d-1$ the sum of the weights of the consistent paths from $\langle s,0\rangle$ to $\langle t,i\rangle$ in C_i (over $\mathcal{K}$) is zero. Thus, C rejects an input u if and only if C_i rejects u for every $0 \le i \le d-1$.

Denote by f_i the function computed by C_i. It follows from the above discussion that $f = \bigvee_{i=0}^{d-1} f_i$. Since we included d^2 edges for each original edge we have $\mathrm{AP}_{\mathcal{K}}(f_i) \le d^2 \cdot \mathrm{AP}_{\mathcal{K}[\alpha]/p(\alpha)}(f)$. Using Corollary 3.5 we get $\mathrm{AP}_{\mathcal{K}}(f) \le 2d \cdot \sum_{i=0}^{d-1} \mathrm{AP}_{\mathcal{K}}(f_i) \le 2d^4 \cdot \mathrm{AP}_{\mathcal{K}[\alpha]/p(\alpha)}(f)$. □

A corollary of Lemma 4.1 is that arithmetic programs over $\mathrm{GF}(q^d)$ can be simulated efficiently by arithmetic programs over $\mathrm{GF}(q)$. This follows since $\mathrm{GF}(q^d) = \mathrm{GF}(q)[\alpha]/p(\alpha)$ where $p(\alpha)$ is an irreducible polynomial over $\mathrm{GF}(q)$ of degree d. A similar result for span programs was proved in [16] (see also [4]). We use Corollary 4.2 in Section 7.

Corollary 4.2 *Let d be a positive integer and q be a prime-power. Then,*

$$\mathrm{AP}_{\mathrm{GF}(q)}(f) \le 2d^4 \cdot \mathrm{AP}_{\mathrm{GF}(q^d)}(f).$$

4.2 Checking If the Rank of a Matrix is Large

In this section we show how to check if the rank of a matrix over a given field $\mathcal{K}$ is at least some integer r. The proof is composed of three parts. We first use a result of Mulmuley [18] which reduces the problem of computing the rank of a matrix, over an arbitrary field, to the problem of computing the characteristic polynomial of a related matrix[3]. Then we use a construction of Mahajan and Vinay [17] that computes the coefficients of the characteristic polynomial (previous constructions can be found in Borodin et al. [7] or Berkowitz [6], however the construction of [17] results in a smaller arithmetic program). The result of [18] uses an extension field of $\mathcal{K}$, and we simulate the arithmetic over the extension field by an arithmetic program over $\mathcal{K}$ using Lemma 4.1.

Lemma 4.3 *Let $\mathcal{K}$ be any field, and r,s be integers such that $r \le s$. Furthermore, let $A(x)$ be an $s \times s$ matrix over $\mathcal{K}$. There exists an arithmetic program of size $O(s^{14})$ which accepts an input u if and only if* $\mathrm{rank}(A(u)) \ge r$.

Proof: Following Mulmuley, we transform the $s \times s$ matrix $A(x)$ over $\mathcal{K}$ to a $2s \times 2s$ 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

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

[3]Borodin et al. [7] were the first to present such reduction over arbitrary fields. The reduction in [7] is randomized while the reduction in [18] is deterministic. For the lemma the reduction of [7] suffices.

For every $u \in \{0,1\}^n$ it holds that $\mathrm{rank}(B(u)) = 2 \cdot \mathrm{rank}(A(u))$, since X has full rank. (Note that the rank of the matrix $A(u)$ over $\mathcal{K}$ equals to its rank over any extension field of $\mathcal{K}$.) Mulmuley [18] proved the following:

Claim 4.4 ([18]) *Let i be the smallest index such that the coefficient of λ^i in the characteristic polynomial of $B(u)$, i.e., $p(\lambda) = \det(\lambda I - B(u))$, is non-zero. Then the rank of the matrix $B(u)$ is $2s - i$.*

Each entry in $B(x)$ has the form $a \cdot \alpha^j$ or $a \cdot x_i^\epsilon \alpha^j$, where x_i^ϵ 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 in α of degree less than $4s^2$, and it suffices to compute the characteristic polynomial modulo α^{4s^2}.

We use a construction of Mahajan and Vinay [17] to check if a given coefficient of the characteristic polynomial is non-zero. Mahajan and Vinay [17] present a parallel algorithm that computes this coefficient and the main stage in their algorithm is constructing a weighted graph corresponding to the matrix. This graph can be interpreted as an arithmetic program which accepts an input u if and only the given coefficient of the characteristic polynomial is zero. We do not describe the construction of [17].

Claim 4.5 ([17]) *Given a $t \times t$ matrix D over any field $\mathcal{K}$ and an integer ℓ, one can construct a weighted directed graph with $O(t^4)$ edges such that the sum of the weights of the paths between two special vertices of the graph is equal to the coefficient of λ^ℓ in the characteristic polynomial of D. (The weight of a path is the product of the weights of the edges appearing in it.)*

The weights assigned to the edges in the Mahajan-Vinay construction are entries of the matrix D or constants from the field $\mathcal{K}$. When applied to a matrix $D(x)$, the construction can be interpreted as an arithmetic program as follows: if an entry of $D(x)$ of the form ax_i^ϵ is assigned to be the weight of an edge in the construction, we let a be the weight and x_i^ϵ be the label of the corresponding edge in the arithmetic program. For edges that get constants from $\mathcal{K}$ as their weight in the Mahajan-Vinay construction we assign the constant 1 label in the arithmetic program. The arithmetic program obtained this way accepts an assignment u if and only if the coefficient of λ^ℓ in the characteristic polynomial of $D(u)$ is non-zero.

We apply Claim 4.5 with respect to $B(x)$ and get an arithmetic program over $\mathcal{K}(\alpha)$ of size $O(s^4)$, where the weights of the edges are polynomials in $\mathcal{K}[\alpha]$. As we remarked before it is enough to compute the coefficient modulo α^{4s^2}, thus using Lemma 4.1, we get an arithmetic program over $\mathcal{K}$ of size $O(s^{12})$.

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 of $\lambda^0, \lambda^1, \ldots, \lambda^{2s-2r}$ in the characteristic polynomial $\det(\lambda I - B(u))$. Thus, we have to compute the disjunction of $2s - 2r + 1 = O(s)$ arithmetic programs of size $O(s^{12})$. By Corollary 3.5, we can compute the disjunction of these arithmetic programs with an arithmetic program of size $O(s^{14})$. □

4.3 Construction of the Arithmetic Program

Finally, we describe the arithmetic program computing f based on the dependency program for $\overline{f}$.

Theorem 4.6 *Let $\mathcal{K}$ be any field. For an arbitrary Boolean 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. 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 non-diagonal entries of $Y(x)$ 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, 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 $M_i 0^s$ (where M_i is the i-th row of M). 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 4.3, which accepts an input u if and only if $\mathrm{rank}(P(u)) \geq s$, computes the desired function f. $\square$

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 [20]. The results of [20] 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 [20] 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 Theorem 4.6, Corollary 3.5 and Theorem B.1.

Corollary 4.7 *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. That is, even if all literals in an arithmetic program are positive, the arithmetic program might compute a non-monotone function. In [20] 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 can be computed by a small arithmetic program then it can also be computed by a small span program; the size of the resulting span program is only twice the size of the arithmetic program.

Before proving the result, we describe the main properties of our construction. The span program we construct has one column for each input $u \in \{0,1\}^n$; the column corresponding to u in M_u is identically 0. (This is similar to canonical span programs defined in [16].) The above property guarantees that the program rejects an input u if the u-th coordinate of the target vector is nonzero. Although we describe a span program with 2^n columns, it is sufficient to keep a maximal linearly independent set of the columns (see Remark 2.4).

We will use the following notation. Let $\langle G, \mu, w, s, t\rangle$ be an arithmetic program that computes f, where $G = (V, E)$ is a directed acyclic graph. For every vertex $a \in V$ define a vector of length 2^n over $\mathcal{K}$, denoted by $\mathbf{f_a}$, such that the u-th coordinate of $\mathbf{f_a}$, 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.

Observation 5.1 *The function computed by the arithmetic program $\langle G, \mu, w, s, a\rangle$ (with target vertex a) rejects an input u if and only if $f_a(u) = 0$. In particular, $f(u) = 0$ 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:

$$(g \cdot x_i^\epsilon)(u) \triangleq \begin{cases} g(u) & \text{if } u_i = \epsilon \\ 0 & \text{if } u_i \neq \epsilon \ . \end{cases}$$

For every edge $e = (a, b) \in E$ with label x_i^ϵ define $\mathbf{f_e} \triangleq \mathbf{f_a} \cdot \mathbf{x_i^\epsilon}$. If the edge is labeled by 1 then $\mathbf{f_e} \triangleq \mathbf{f_a}$. Let $H_b \subseteq E$ denote the set of all edges whose head is the vertex $b \in V$. Then the following equation holds

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

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

Our construction will maintain the property that, for every input u and every vertex a, the vector $\mathbf{f_a} - f_a(u)\mathbf{1}$ is spanned by the rows of M_u. For example, over GF(2) our construction would have the property that if the number of paths between s and a that are consistent with u is even then $\mathbf{f_a}$ is spanned by the rows of M_u, otherwise $\mathbf{f_a} - \mathbf{1}$ is spanned by the rows of M_u.

Theorem 5.2 *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 $\widehat{M}$ (over $\mathcal{K}$) that computes $\overline{f}$, and has size not more than twice the size of the arithmetic program. Since $\mathrm{SP}_{\mathcal{K}}(f) = \mathrm{SP}_{\mathcal{K}}(\overline{f})$ (see [11, 20]) this suffices.

We next describe the span program $\widehat{M}$. For every edge $e = (a, b)$ labeled by a literal x_i^ϵ, 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 we prove that if $\overline{f}(u) = 0$ then $\mathbf{f_t} \notin \mathrm{span}(M_u)$. By Observation 5.1 if $\overline{f}(u) = 0$ then $f_t(u) \neq 0$. Thus, the u-th coordinate in $\mathbf{f_t}$ is not zero. Since our span program has the property that on every input u the column corresponding to u in M_u is identically 0, the program rejects u if the u-th coordinate in $\mathbf{f_t}$ is not zero, as claimed.

Next we prove that $\overline{f}(u) = 1$ implies that $\mathbf{f_t} \in \mathrm{span}(M_u)$. This will follow from the following claims.

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

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

By Observation 5.1 if $\overline{f}(u) = 1$ then $f_t(u) = 0$. 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 a topological 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^ϵ. 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 $x_i^{u_i}(u) = 1$ (that is, u satisfies $x_i^{u_i}$), 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 $\mathbf{f_a} - f_a(u)\mathbf{1} \in \mathrm{span}(M_u)$, and by the construction of $\widehat{M}$ it holds that $\mathbf{f_a} \cdot \mathbf{x_i^{\overline{\epsilon}}} \in \mathrm{span}(M_u)$, 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 $\mathrm{span}(M_u)$. So, $\mathbf{f_b} - f_b(u)\mathbf{1}$ is also in $\mathrm{span}(M_u)$. This completes the proof of both claims. We have proved that the span program $\widehat{M}$ accepts exactly those inputs that satisfy $\overline{f}$, as desired. □

Theorem 5.2 is a generalization of a result of Karchmer and Wigderson [16], 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 5.2 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 $2n$ (each literal has to label at least one row since the value of the function can change from 1 to 0 both when any given variable is changed from 0 to 1 and when it is changed from 1 to 0).

6 Simulating Dependency Programs by Span Programs

In this section we continue the discussion on the connection between the algebraic models considered in the paper, and prove that over "large" fields span programs can simulate dependency programs without any overhead. We also show that even for small fields the overhead of the simulation can be limited to a multiplicative factor of n, that is $\mathrm{SP}_{\mathcal{K}}(f) \le n \cdot \mathrm{DP}_{\mathcal{K}}(f)$ for every field $\mathcal{K}$. Note that our constructions are non-uniform. This should be compared to [20], where Pudlák and Sgall proved that $\mathrm{SP}_{\mathcal{K}}(f) \le (\mathrm{DP}_{\mathcal{K}}(f))^2$ providing a uniform construction.

Lemma 6.1 *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function and $\mathcal{K}$ be a field such that $|\mathcal{K}| \ge 2^n$. Then, $\mathrm{SP}_{\mathcal{K}}(f) \le \mathrm{DP}_{\mathcal{K}}(f)$.*

Proof: Let $\widehat{M} = \langle M, \rho \rangle$ be a dependency program over $\mathcal{K}$ that computes f, and denote the number of rows of M (that is, the size of $\widehat{M}$) by s. Let $\mathbf{r}$ be a column vector with s coordinates whose values will be specified later. Define the span program $\widehat{M}' = \langle M', \rho', \mathbf{v} \rangle$ where $\rho = \rho'$, $\mathbf{v} = \langle 1, 0, \ldots, 0 \rangle$, and $M' = (\mathbf{r}, M)$, that is, the new column $\mathbf{r}$ is added before the columns of M. We will prove that with an appropriate choice of $\mathbf{r}$ the span program $\widehat{M}'$ computes the same function as the dependency program $\widehat{M}$.

We first claim that every input accepted by $\widehat{M}'$ is accepted by $\widehat{M}$ (this claim is independent of the choice of $\mathbf{r}$). Let u be an input that $\widehat{M}'$ accepts. There exist coefficients $\alpha_1, \ldots, \alpha_s$ such that $\sum_{j=1}^{s} \alpha_j M'_j = \langle 1, 0, \ldots, 0 \rangle$ and each α_j is non-zero only in rows of M_u (here, M'_j is the j-th row of M'). Clearly, there exists at least one α_j which is non-zero. By the construction of $\widehat{M}'$, we have $\sum_{j=1}^{s} \alpha_j M_j = \langle 0, \ldots, 0 \rangle$, and the dependency program $\widehat{M}$ accepts u.

Now, let u be an input that $\widehat{M}$ accepts. There exists coefficients $\alpha_1, \ldots, \alpha_s$ such that $\sum \alpha_j M_j = \langle 0, \ldots, 0 \rangle$ where α_j is non-zero only in rows of M_u and there exists at least one α_j which is non-zero. Now, $\sum_{j=1}^{s} \alpha_j M'_j = \langle \sum_{j=1}^{s} \alpha_j \cdot r_j, 0, \ldots, 0 \rangle$. Thus, if

$$\sum_{j=1}^{s} \alpha_j \cdot r_j \neq 0 \tag{5}$$

then $\widehat{M}'$ accepts u.

Let $A \subseteq \mathcal{K}$ be any subset of size 2^n. We analyze the number of vectors $\mathbf{r}$ from A^s that violate (5) for the fixed u. Let i be a coordinate such that $\alpha_i \neq 0$. For every vector from A^{s-1} assigned to the coordinates r_j of $\mathbf{r}$ for $j \neq i$ there exists at most one value for r_i such that $\sum_{j=1}^{s} \alpha_j \cdot r_j = 0$. Thus, there are at most $|A|^{s-1}$ vectors in A^s that violate (5) for a fixed input u. There are at most $2^n - 1$ inputs such that $f(u) = 1$ (otherwise $f \equiv 1$ and the lemma is trivial). Therefore there are at most $(2^n - 1) \cdot |A|^{s-1} < |A|^s$ vectors that might cause $\widehat{M}'$ to reject an input that $\widehat{M}$ accepts. To summarize, there is at least one vector $\mathbf{r}$ in A^s for which $\widehat{M}'$ accepts an input if and only if $\widehat{M}$ accepts the input. □

The previous simulation does not increase the size of the program and it preserves monotonicity: If all the labels of the rows of the dependency program are variables (and not negated variables) then the resulting span program is monotone as well. However, this simulation works only over "large" fields.

We next show that span programs can simulate dependency programs with an overhead limited to a multiplicative factor of n over every field.

Corollary 6.2 *Let f be a Boolean function and $\mathcal{K}$ be any field. Then, $\mathrm{SP}_{\mathcal{K}}(f) \le n \cdot \mathrm{DP}_{\mathcal{K}}(f)$.*

Proof: If $\mathcal{K}$ contains at least 2^n elements then we use Lemma 6.1. Otherwise, $\mathcal{K} = \mathrm{GF}(q)$ for some prime-power $q < 2^n$, and we apply Lemma 6.1 over $\mathrm{GF}(q^n)$:

$$\mathrm{SP}_{\mathrm{GF}(q^n)}(f) \leq \mathrm{DP}_{\mathrm{GF}(q^n)}(f) \leq \mathrm{DP}_{\mathrm{GF}(q)}(f)$$

(the last inequality holds since every dependency program over $\mathrm{GF}(q)$ is also a dependency program over $\mathrm{GF}(q^n)$). To complete the proof we use the following result, proved in [16]: $\mathrm{SP}_{\mathrm{GF}(q)}(f) \leq n \cdot \mathrm{SP}_{\mathrm{GF}(q^n)}(f)$. □

7 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 [25] proved 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 [25] directly generalizes to arithmetic programs over arbitrary fields, and yields arithmetic programs of size $O\left(n|V|^2\left(\mathrm{NBP}(f)\right)^3\right)$, where n is the number of variables, and $|V|$ and $\mathrm{NBP}(f)$ are the number of vertices and the size, respectively, of the smallest nondeterministic branching program computing f. We give an alternative construction which is more efficient, i.e., the resulting arithmetic program is smaller (see Theorem 7.3).

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 [21]. In this paper a nondeterministic branching program is a quadruple $\langle G, \mu, s, t\rangle$ where G is a directed acyclic graph, μ 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. The size of the smallest nondeterministic branching program that computes f is denoted by $\mathrm{NBP}(f)$.

A simple idea to simulate nondeterministic branching programs by arithmetic programs is to give 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 large enough, but still not too large, 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 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 using Corollary 4.2. We now formalize these ideas.

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 on the path, and denote the sum of the weights of all directed paths from s to a in G by $\mathrm{wt}_w(G, a)$. We use the convention $\mathrm{wt}_w(G, s) = 1$.

Lemma 7.1 *If there is a path from s to t in G then $\Pr[\mathrm{wt}_w(G, t) = 0] < |V|/|A|$, where w is chosen uniformly from the space of functions from E to A.*

Proof: We prove, by induction, that for every $a \in V$ if the length of the shortest path from s to a is ℓ then $\Pr[\mathrm{wt}_w(G, a) = 0] \leq \ell/|A|$. The basis is trivial since $\mathrm{wt}_w(G, s) = 1$. 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 β_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 $\mathrm{wt}_w(G, a)$ and β_w are independent of $w((a, b))$. Using this notation $\mathrm{wt}_w(G, b) = \mathrm{wt}_w(G, a) \cdot w((a, b)) + \beta_w$. Thus, if $\mathrm{wt}_w(G, a) \neq 0$ then there is exactly one value of $w((a, b))$ such that $\mathrm{wt}_w(G, b) = 0$. By the induction hypothesis, the probability that $\mathrm{wt}_w(G, a) = 0$ is at most $\ell/|A|$ and the probability that $w((a, b)) = -\beta_w/\mathrm{wt}_w(G, a)$ is at most $1/|A|$. $\square$

The next lemma shows how to construct an arithmetic program simulating a nondeterministic branching program over sufficiently large fields. A special case of the lemma is that if the field contains at least 2^{2n} elements then $\mathrm{AP}_{\mathcal{K}}(f) \leq \mathrm{NBP}(f)$.

Lemma 7.2 *Let T be an integer, $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function, and $\mathcal{K}$ be a field with at least $T \cdot \mathrm{NBP}(f)$ elements. 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 $T \cdot \mathrm{NBP}(f)$. 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, $\mathrm{wt}_w(G(u), t) = 0$, 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 7.1, the probability that $\mathrm{wt}_w(G(u), t) = 0$ 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 $2^n \cdot T^{-k} \leq 1$. Thus, there exist k weight functions such that $f = \bigvee_{j=1}^{k} g_j$, and $\mathrm{AP}_{\mathcal{K}}(f) = O\left(k \sum_{j=1}^{k} \mathrm{AP}_{\mathcal{K}}(g_j)\right) = O\left(k^2 \cdot \mathrm{NBP}(f)\right)$. $\square$

We next show how to simulate non-deterministic branching programs by arithmetic programs over small fields.

Theorem 7.3 *For any field $\mathcal{K}$ and an arbitrary Boolean function f in n variables*

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

Proof: If the field $\mathcal{K}$ contains at least $(\mathrm{NBP}(f))^2$ elements then the theorem follows from Lemma 7.2. Otherwise, use a finite extension of $\mathcal{K}$ which contains at least $(\mathrm{NBP}(f))^2$ elements. By Lemma 7.2, the function f can be computed over such extension of $\mathcal{K}$ by an arithmetic program of size

$$O\left(\frac{n^2 \cdot \mathrm{NBP}(f)}{(\log \mathrm{NBP}(f))^2}\right).$$

Now, apply Corollary 4.2 to get an arithmetic program over $\mathcal{K}$. □

Over GF(2) we can reduce the size of the arithmetic program to

$$O\left(n \cdot \mathrm{NBP}(f) \cdot (\log \mathrm{NBP}(f))^2\right).$$

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). We use this fact to improve the simulations over GF(2) in Corollary 4.2 and in Lemma 7.2.

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 7.2 over an extension field with at least 2^{2n} elements by a span program using Theorem 5.2. The span program over smaller fields is obtained by the result of [16] stating that $\mathrm{SP}_{\mathrm{GF}(q)}(f) \le d \cdot \mathrm{SP}_{\mathrm{GF}(q^d)}(f)$.

8 Conclusions and Open Problems

In this work we considered three algebraic models of computation: arithmetic branching programs, span programs and dependency programs. In addition we considered nondeterministic branching programs. We next summarize the connections between these models. For a function $f : \{0,1\}^n \to \{0,1\}$

$$\frac{\mathrm{SP}_{\mathcal{K}}(f)}{n} = \frac{\mathrm{SP}_{\mathcal{K}}\left(\overline{f}\right)}{n} \le \mathrm{DP}_{\mathcal{K}}\left(\overline{f}\right) \le 2 \cdot \mathrm{AP}_{\mathcal{K}}(f) \le O\left(n^2 \cdot \mathrm{NBP}(f)(\log \mathrm{NBP}(f))^2\right).$$

Furthermore, for some constant c

$$\mathrm{AP}_{\mathcal{K}}(f) \le O\left(\mathrm{DP}_{\mathcal{K}}\left(\overline{f}\right)^c\right).$$

Over infinite fields (and fields that contain at least 2^{2n} elements) the connections are tighter

$$\mathrm{SP}_{\mathcal{K}}(f) = \mathrm{SP}_{\mathcal{K}}\left(\overline{f}\right) \le \mathrm{DP}_{\mathcal{K}}\left(\overline{f}\right) \le 2 \cdot \mathrm{AP}_{\mathcal{K}}(f) \le 2 \cdot \mathrm{NBP}(f).$$

We mention three open problems that arise from our work.

Open Problem 1 *Do span programs and arithmetic programs have the same power over every field? That is, does there exist a constant c such that* $\mathrm{AP}_{\mathcal{K}}(f) = O(\mathrm{SP}_{\mathcal{K}}(f)^c)$ *for every field* $\mathcal{K}$*?*

Recall that span programs and arithmetic programs are equivalent over fixed finite fields up to polynomial factors. The size of the simulation of span programs by arithmetic programs depends on the number of elements in the field, that is $\mathrm{AP}_{\mathrm{GF}(q)}(f) = O\left(q \cdot \mathrm{SP}_{\mathrm{GF}(q)}(f)\right)^{O(1)}$.

Open Problem 2 *Are arithmetic programs closed under complement 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 complement over the reals or rationals?*

Notice that for every finite field $\mathrm{AP}_{\mathrm{GF}(q)}\left(\overline{f}\right) \leq (q-1) \cdot \mathrm{AP}_{\mathrm{GF}(q)}(f)$.

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* $\mathrm{rank}(A(u)) = r$*?*

We claim that the three open problems are equivalent. If arithmetic programs are closed under complement then, by Lemma 4.3, there is an arithmetic program of size $s^{O(1)}$ that checks if the rank of an $s \times s$ matrix $A(u)$ is less than r. Hence, an arithmetic program to check if $\mathrm{rank}(A(u)) = r$ can be obtained as the conjunction of two polynomial size arithmetic programs (checking if the rank is at least r and less than $r+1$). Thus, a positive answer to Open Problem 2 implies a positive answer to Open Problem 3.

Recall that a span program $\widehat{M} = \langle M, \rho, \mathbf{v}\rangle$ accepts an input u if and only if $\mathbf{v} \in \mathrm{span}(M_u)$ which is true if and only if

$$\mathrm{rank}(M_u \cup \{\mathbf{v}\}) = \mathrm{rank}(M_u).$$

Now, define matrices $A(x)$ and $B(x)$ where $A(x)_{i,j} = M_{i,j} \cdot \rho(i)$, and $B(x)$ is obtained from $A(x)$ by adding the vector $\mathbf{v}$ as an extra row. Thus, $\widehat{M}$ accepts u if and only if

$$\exists r,\ \mathrm{rank}(A(u)) = r \text{ and } \mathrm{rank}(B(u)) = r. \tag{6}$$

If the answer to Open Problem 3 is positive then, by the closure properties of arithmetic programs, there is an arithmetic program accepting an input u if and only if (6) is true, and this arithmetic program computes the same function as $\widehat{M}$. Thus, a positive answer to Open Problem 3 implies a positive answer to Open Problem 1.

Finally, if arithmetic programs are equivalent to span programs then they are closed under complement since span programs are closed under complement [11, 20]. That is, given an arithmetic program for f we simulate it by a span program for f, next we construct a span program for $\overline{f}$, and then we simulate the span program for $\overline{f}$ by an arithmetic program for $\overline{f}$. 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] asking if the complexity class $\mathcal{C}_{=}\mathcal{L}$ is closed under complement (see also the Introduction).

Acknowledgments

We would like to thank Eric Allender, Allan Borodin, Eyal Kushilevitz, and Avi Wigderson for helpful discussions, and the Computational Complexity 98 program committee and the anonymous referee for many helpful comments.

References

[1] E. Allender. Making computation count: Arithmetic circuits in the nineties. *SIGACT NEWS, Complexity Theory Column*, 28(4):2–15, 1997.

[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. Technical Report 96-37, DIMACS, 1996. To appear in *Combinatorica*.

[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] A. Borodin, J. von zur Gathen, and J. Hopcroft. Fast parallel matrix and GCD computations. *Information and Control*, 52(3):241–256, 1982.

[8] A. Borodin, A. A. Razborov, and R. Smolensky. On lower bounds for read-k-times branching programs. *Computational Complexity*, 3(1):1–18, 1993.

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

[10] 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/, 1997.

[11] A. Gál. *Combinatorial Methods in Boolean Function Complexity*. PhD thesis, U. of Chicago, 1995. Also: http://www.eccc.uni-trier.de/eccc-local/ECCC-Theses/gal.html .

[12] A. Gál. A characterization of span program size and improved lower bounds for monotone span programs. In *Proc. of the 30th Annu. ACM Symp. on the Theory of Computing*, pages 429–437, 1998.

[13] J. von zur Gathen. Algebraic complexity theory. *Ann. Review of Comp. Sci.*, 3:317–347, 1988.

[14] J. von zur Gathen. Parallel linear algebra. In J. H. Reif, editor, *Synthesis of Parallel Algorithms*, pages 573–617. Morgan Kaufmann, 1993.

[15] 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.

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

[17] M. Mahajan and V Vinay. Determinant: Combinatorics, algorithm, and complexity. *Chicago J. of Theoretical Comp. Sci.,* http://www.cs.uchicago.edu/publications/cjtcs/, 1997:5, 1997. Preliminary version: A combinatorial algorithm for the determinant, *Proc. of the 8th Annu. ACM-SIAM Symp. on Discrete Algorithms*, pages 730–738, 1997.

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

[19] 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.

[20] P. Pudlák and J. Sgall. Algebraic models of computation and interpolation for algebraic proof systems. In P. W. Beame and S. Buss, editors, *Proof Complexity and Feasible Arithmetic*, volume 39 of *DIMACS Series in Discrete Mathematics and Theor. Comp. Sci.*, pages 279–296. AMS, 1998.

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

[22] K. Reinhardt and E. Allender. Making nondeterminism unambiguous. In *Proc. of the 38th Annu. IEEE Symp. on Foundations of Computer Science*, pages 244–253, 1997.

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

[24] 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.

[25] 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 & Algorithms*, 9:99–111, 1996. See also: ECCC report TR95-049, http://www.eccc.uni-trier.de/eccc/, 1995.

A Irreducible Polynomials over Finite Fields

To construct an arithmetic program over finite fields for a disjunction of two functions, we use irreducible polynomials. 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., [23]), and it can be derived by simple counting arguments. In this appendix we include, for completeness, a constructive proof of this result (see [23]). For this proof we consider two cases: 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 $\mathrm{GF}(q)$, 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* $\mathrm{GF}(q)$*. The polynomial $x^2 - a$ is irreducible over* $\mathrm{GF}(q)$*.*

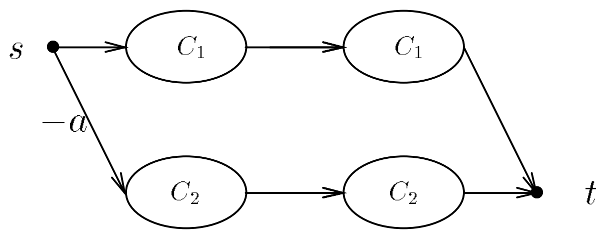

Figure 5: An arithmetic program for $f_1 \vee f_2$, where a is a quadratic non-residue.

Hence $x^2 - ay^2 = 0$ if and only if $x = y = 0$ and the arithmetic program for $f_1 \vee f_2$ over finite fields of odd characteristic, as proved in Lemma 3.3, is described in Fig. 5.

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

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

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

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 $\mathrm{GF}(2^i)$ 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, $a = b^2 + b$. Using the fact that the characteristic of the field is two, we get

$$\begin{aligned} T(a) &= \sum_{j=0}^{i-1} a^{2^j} = \sum_{j=0}^{i-1} (b^2 + b)^{2^j} \\ &= \sum_{j=0}^{i-1} \left((b^2)^{2^j} + b^{2^j} \right) = \sum_{j=0}^{i-1} b^{2^{j+1}} + \sum_{j=0}^{i-1} b^{2^j} \\ &= b^{2^i} + b = b + b = 0. \end{aligned}$$

But a was chosen such that $T(a) \neq 0$, and $x^2 + x + a$ is irreducible over $\mathrm{GF}(2^i)$. $\square$

Hence $x^2 + xy + ay^2 = 0$ if and only if $x = y = 0$ and the arithmetic program for $f_1 \vee f_2$ over $\mathrm{GF}(2^i)$, as proved in Lemma 3.3, is described in Fig. 6.

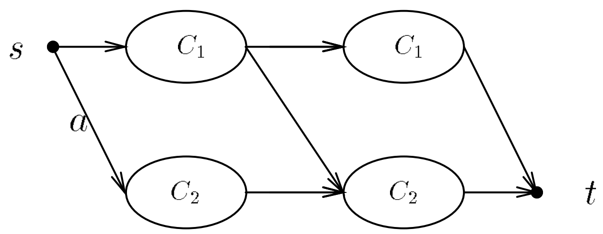

Figure 6: An arithmetic program over a finite field of characteristic two for $f_1 \vee f_2$, where $T(a) \neq 0$.

B Simulating Complements of Arithmetic Programs by Dependency Programs

In this section we show how to construct a dependency program computing $\overline{f}$ from an arithmetic program computing f, while only doubling the size. The proof is based on the arguments of [20].

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 if the arithmetic program rejects the input u. This part of the argument is a straightforward generalization of the arguments of [20] to arithmetic programs. Next, we transform this matrix into a dependency program, similarly to the proof in [20], 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 $\alpha \cdot x_i^\epsilon$ or α (where α is an element in $\mathcal{K}$). We first describe an intermediate matrix $A(x)$ whose rows and columns are indexed by vertices of G ordered topologically from s to t (that is, if there is a path from a to b in G then a appears before b). 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 row (indexed by t). The following lemma is proved in [9] for mod-p branching programs, but the proof applies to arithmetic programs with arbitrary weights. For sake of completeness, we present a proof of the lemma. (We present a proof which is somewhat different from the original proof of [9] and uses ideas of [15].)

Lemma B.2 ([9]) *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).*

Proof: The intuition behind the lemma is that the determinant $\det(B(u))$ is a sum of terms where each term is the weight of a path in G from s to t that is consistent with u. Formally, for every $u \in \{0,1\}^n$ we define the column vector $\mathbf{d}$ such that d_a is equal to the sum of the weights of the paths between a and t that are consistent with u (for example, $d_s = C(u)$ and $d_t = 1$). We show that $\mathbf{d}$ is a solution to a system of linear equations.

$$(I - A(u))\mathbf{d} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}. \tag{7}$$

That is, we show that for every vertex a if we denote the a-th row of $A(u)$ by $A(u)_a$ then

$$A(u)_a \cdot \mathbf{d} = \begin{cases} d_a & \text{if } a \neq t \\ d_t - 1 = 0 & \text{if } a = t. \end{cases}$$

By definition, $A(u)_a \cdot \mathbf{d}$ is equal to the sum of $w(a,c)d_c$ over all c such that the label of (a,c) is consistent with u. Since G is acyclic, whenever $a \neq t$ this sum equals d_a, and when $a = t$ this sum is 0. Thus, Equation (7) holds. By Cramer's rule we get that

$$d_s = \pm \frac{\det(B(u))}{\det(A(u) - I)} = \pm \det(B(u)).$$

($\det(A(u) - I) = \pm 1$ since $A(u) - I$ is a triangular matrix with -1's on its main diagonal.) $\Box$

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 variables in each row (that is, entries $\alpha_1 \cdot x_{i_1}^{\epsilon_1}$ and $\alpha_2 \cdot x_{i_2}^{\epsilon_2}$ for $i_1 \neq i_2$). 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 (thus, the size of the arithmetic program) 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^ϵ 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. In the second row, labeled by x_i^ϵ, for each entry of the form $\alpha \cdot x_i^\epsilon$ or α the corresponding coordinate is α.

We constructed a dependency program $\widehat{M}$ computing $\overline{f}$. The only problem is that this program contains rows labeled by 1 which is not allowed by our definition. As Pudlák and Sgall observed in [20], one can transform $\widehat{M}$ to a dependency program computing the same function without rows labeled by 1. For completeness, we describe their transformation. Without loss of generality, let us assume that f is not constant (otherwise the statement of the theorem is trivial). We apply a linear transformation to the rows of $\widehat{M}$ such that the null space of the linear transformation is the linear space spanned by the rows labeled by 1 in $\widehat{M}$. The result of the transformation is a dependency program which computes the same function and has no rows labeled by 1. The size of the transformed dependency program is at most the number of rows labeled by a literal in $\widehat{M}$, which in turn is at most twice the number of edges in the arithmetic program computing f. We described the dependency program computing $\overline{f}$, which completes the proof of the theorem. $\square$