Electronic Colloquium on Computational Complexity, Report No. 48 (2008)

Arithmetic circuits, syntactic multilinearity, and the limitations of skew formulae

Meena Mahajan and B. V. Raghavendra Rao

The Institute of Mathematical Sciences, Chennai 600 113, India. {meena,bvrr}@imsc.res.in

Abstract. Functions in arithmetic NC^1 are known to have equivalent constant width polynomial degree circuits, but the converse containment is unknown. In a partial answer to this question, we show that syntactic multilinear circuits of constant width and polynomial degree can be depth-reduced, though the resulting circuits need not be syntactic multilinear. We then focus specifically on polynomial-size syntactic multilinear circuits, and study relationships between classes of functions obtained by imposing various resource (width, depth, degree) restrictions on these circuits. Along the way, we obtain a characterisation of NC^1 (and its arithmetic counterparts) in terms of log width restricted planar branching programs. We also study the power of skew formulae, and show that even exponential sums of these are unlikely to suffice to express the determinant function.

1 Introduction

Among the parallel complexity classes, the class NC^1 of boolean functions computed by logarithmic depth polynomial size circuits has several equivalent characterisations, in the form of bounded width branching programs, polynomial size formulae and bounded width circuits of polynomial size. Its subclass AC^0, consisting of polynomial size constant depth unbounded fan-in circuits, has also been characterised via restricted branching programs. See Figure 1.

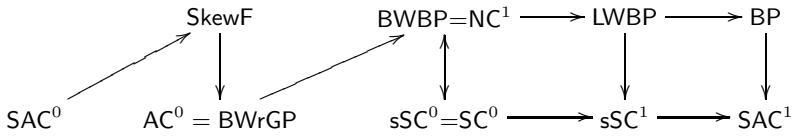

Fig. 1. Boolean complexity classes around NC^1

However, when we consider the counting and arithmetic versions of those classes which are equivalent to NC^1, they seem to represent different classes of functions. In [12], it was shown that counting the total weights of paths over $\mathbb{Z}$ in a bounded width branching program is equivalent to the functions computable by log depth polynomial size arithmetic circuits over $\mathbb{Z}$, *i.e.* $\mathsf{GapBWBP} = \mathsf{GapNC}^1$. In [14], this study was extended to bounded width circuits of polynomial degree and size, *i.e.* sSC^0, showing that $\mathsf{GapNC}^1 \subseteq \mathsf{GapsSC}^0$, but it left open the question of equality of these classes.

The question of whether GapsSC^0 is in GapNC^1 can be seen as a depth reduction problem for bounded width circuits. We do not have an answer for this general question. So it is natural to ask if there are any restrictions on the circuit so that depth reduction is possible.

ISSN 1433-8092

Syntactic multilinearity is a restriction which has been studied in the literature. Syntactic multilinear circuits are those in which every multiplication gate operates on disjoint set of variables. Obviously, the polynomials computed by syntactic multilinear circuits are multilinear. The syntactic multilinear restriction is very fruitful in the sense that there are known unconditional separations and lower bounds for these classes (see [17–19]).

We show that depth reduction for small width circuits is possible if the circuit is syntactic multilinear; however, the depth-reduced circuit may not be syntactic multilinear or even multilinear. The setting we consider is more general than that of [12] and [14]; here the input variables are allowed to take arbitrary values from the underlying ring $\mathbb{K}$. The main result is that polynomial size, constant width syntactic multilinear circuits can be simulated (non-uniformly) by log depth bounded fan-in circuits of polynomial size, but this construction need not preserve the syntactic multilinearity property.

Once we take up the restriction of syntactic multilinearity for these arithmetic circuits, it is worthwhile to explore the relationships among the syntactic multilinear arithmetic circuit classes close to arithmetic NC^1.

In the model of branching programs, syntactic multilinearity is a well-studied notion and it is referred to as read-once branching programs (see [8] for example). There are several known lower bounds for syntactic multilinear branching programs. (see *e.g.* [7, 6]). For formulae, syntactic multilinearity is defined exactly as for circuits. A careful observation of the depth reduction for poly size arithmetic formula as given in [9] shows that it preserves syntactic multilinearity. Also some of the constructions in [12–14], relating branching programs and formulae, can be shown to preserve syntactic multilinearity.

In [3], the class of bounded depth arithmetic circuits is characterised in terms of a restricted version of grid programs, rGP, of bounded width BWrGP. We observe that this construction can be extended to show a new (non-uniform) characterisation of arithmetic NC^1 in terms of restricted planar branching programs of log width LWrGP. In addition, this can be shown to preserve syntactic multilinearity, for arithmetic NC^1 as well as arithmetic AC^0.

We also study the class of polynomial size skew formulas, denoted SkewF. The motivation for this study arises from Valiant's characterisations of the classes VP and VNP (see [22]; also, for more exposure on algebraic complexity theory, the reader is referred to [10, 11]). Valiant proved that every polynomial $p(X) \in \mathsf{VNP}_{\mathbb{K}}$ (where $\mathbb{K}$ is an arbitrary ring), and in particular every polynomial in $\mathsf{VP}_{\mathbb{K}}$, can be written as $p(X) = \sum_{e \in \{0,1\}^m} \phi(X, e)$, where the polynomial ϕ has an arithmetic formula of polynomial size. We know that the class of "Permanent" (see *e.g.* [10]) polynomials is complete for VNP. It is also known [21] that the class "Determinant" is equivalent to skew circuits of polynomial size. The question we ask is: can we prove a similar equivalence in the case of skew circuits? That is, can we write polynomials computed by skew circuits as an exponential sum of polynomials computed by skew formulae? We show that this is highly unlikely, by showing that any polynomial which is expressible as an exponential sum of skew formulae belongs to the class VNC^1.

The existing and new relationships amongst the arithmetic classes (prefix a-) can be seen in Figure 2; Figure 3 shows the corresponding picture for the syntactic multilinear classes

(prefix sma-). Note that our main depth-reduction result straddles the two figures, and along with [14] gives sma-sSC$^0 \subseteq$ a-NC$^1 \subseteq$ a-sSC^0.

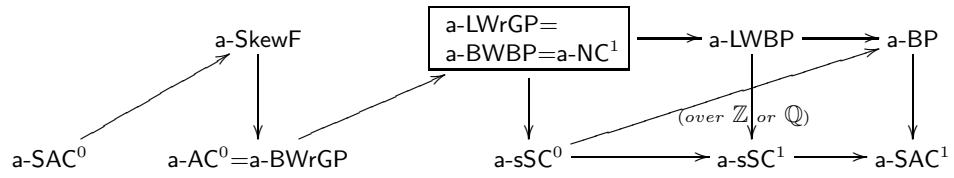

Fig. 2. Arithmetic classes around NC^1

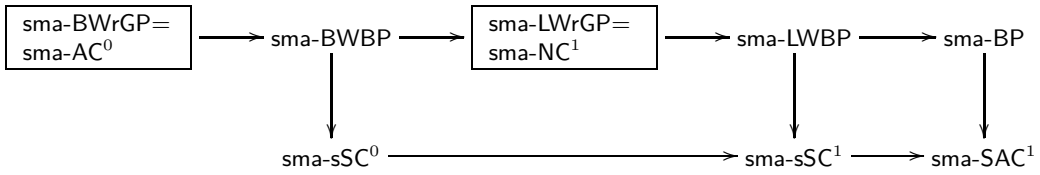

Fig. 3. Relationship among syntactic multilinear classes

The rest of the paper is organised as follows. Section 2 introduces basic definitions. In Section 3 we prove that small-width syntactic multilinear circuits can be depth-reduced. In Section 4, we establish the containments among the syntactic multilinear classes and obtain a new characterisation for NC^1 in terms of a restricted class of grid branching programs. In Section 5 we describe our results concerning skew formulae.

2 Preliminaries

Boolean circuit classes: A boolean circuit is a directed acyclic graph, where nodes are labelled by $\{0, 1, \wedge, \vee, x_1, \ldots, x_n, \neg\}$, where nodes with label from $\{0, 1, x_1, \ldots, x_n\}$ are circuit inputs, and designated nodes of zero out-degree are called the *output* gates. The *fan-in* (*fan-out*) of a gate is its in-degree (out-degree). The size, width, depth and degree of a circuit are defined in the standard sense (*e.g.* , see [14],[25]). Unless otherwise stated, fan-in is assumed to be bounded. NC^1 denotes the class of boolean functions $f_n : \{0,1\}^n \to \{0,1\}$ which can be computed by boolean circuits of depth $O(\log n)$ and size poly(n). SC^i denotes the class of functions computed by polynomial size circuits of width $O(\log^i n)$. sSC^i is the class of boolean functions computed by polynomial degree, polynomial size circuits of width $O(\log^i n)$. SAC^i denotes the class of boolean functions computed by polynomial circuits of size poly(n) and depth $O(\log^i n)$, where $\vee$ gates can have unbounded fan-in. AC^0 denotes the class of boolean functions which can be computed by unbounded fan-in constant depth boolean circuits of size poly(n).

A *formula* is a circuit where every non-input gate has fan-out bounded by one. F denotes the set of boolean functions which can be computed by polynomial size formulae. LWF

denotes the class of functions computed by boolean formulae of log width and polynomial size. Without loss of generality, NC^1, AC^0 and SAC^0 circuits can be assumed to be formulae.

Branching programs: A branching program (BP) is a directed acyclic graph (usually layered) with two designated nodes s and t. Edges in this graph are labelled by literals from $\{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n, 0, 1\}$, where $x_i \in \{0, 1\}$ are the set of inputs. A BP is said to accept its input if and only if there exists a s-t path, in which every edge label evaluates to 1. A BP can also be viewed as a skew-circuit, *i.e.* a circuit where every $\wedge$ gate has at most one non-circuit input. Let BWBP and LWBP denote the functions computed by constant width and log width branching programs of polynomial size respectively.

G-graphs are the graphs that have planar embeddings where vertices are embedded on a rectangular grid, and all edges are between adjacent columns from left to right. Let BWGP denote the class of boolean functions accepted by constant width polynomial size branching programs which are G-graphs. In the above graph, the node s is fixed as the leftmost bottom node and t is the rightmost top node. In [3], a restriction of G-graphs is considered where the width of the grid is a constant, and only certain kinds of connections are allowed between any two layers. Namely, for width $2k+2$, the connecting pattern at any layer is represented by one of the graphs $G_{k,i}$ (see figure 2) for $0 \leq i \leq 2k+2$. Let BWrGP denote the class of boolean functions accepted by constant width polynomial size branching programs that are restricted G-graphs, and LWrGP the class corresponding to log width polynomial size programs that are restricted G-graphs. (see [3]).

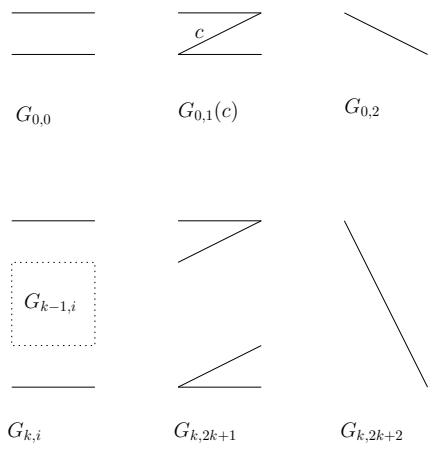

Fig. 4. The possible patterns between two layers of rGPs

The following proposition summaries the known relationships among the boolean complexity classes defined above; see for instance [25].

Proposition 1 ([3, 4, 20, 13, 24]). *The following results are known.*

$$\mathsf{AC}^0 = \mathsf{BWrGP}$$

$$\mathsf{NC}^1 = \mathsf{BWBP} = \mathsf{SC}^0 = \mathsf{sSC}^0 = \mathsf{F} = \mathsf{LWF}$$
$$\mathsf{SAC}^1 = \text{Circuit Size, } \mathsf{Deg}(\mathsf{poly}(n), \mathsf{poly}(n))$$

Arithmetic and counting classes: An arithmetic circuit over a ring $\langle \mathbb{K}, +, -, \times, 0, 1 \rangle$ is a circuit where the nodes are labelled from $\{\times, +\}$ and constants from $\mathbb{K}$ along with input variables $x_1, \ldots, x_n$ that take values in $\mathbb{K}$. The constants $\{-1, 0, 1\}$ are assumed to be the only constants from $\mathbb{K}$ available free of cost. The degree of a node f is inductively defined as follows: the degree of constants and circuit input variables is 1. If $f = g \times h$ then $\deg(f) = \deg(g) + \deg(h)$, and if $f = g + h$ then $\deg(f) = \max\{\deg(g), \deg(h)\}$. The degree of a circuit is the degree of its output node. Note that the degree of the polynomial computed by an arithmetic circuit is bounded by the degree of the circuit.

The arithmetic circuit classes corresponding to the above defined boolean classes consist of functions $f : \mathbb{K}^* \to \mathbb{K}$, and are defined as follows.

$$\mathsf{BWBP}[\mathbb{K}] = \{f : \mathbb{K}^* \to \mathbb{K} \mid f = \text{sum of weights of all } s \rightsquigarrow t \text{ paths in a } \mathsf{BWBP}\}$$

Here the weight of a path is the product of the labels of edges along the path.

$$\mathsf{NC}^1[\mathbb{K}] = \left\{ f \;\middle|\; \begin{array}{l} f \text{ has a poly size, } O(\log n) \text{ depth, fan-in 2} \\ \text{circuit.} \end{array} \right\}$$

$$\mathsf{sSC}^i[\mathbb{K}] = \left\{ f \;\middle|\; \begin{array}{l} f \text{ has a poly size, } O(\log^i n) \text{ width, } \mathsf{poly}(n) \text{ degree} \\ \text{circuit.} \end{array} \right\}$$

For notational convenience we drop the $\mathbb{K}$ where understood from context to be any (or a specific) ring. To distinguish from the boolean case, we write $\mathcal{C}[\mathbb{K}]$ as $\mathsf{a}\text{-}\mathcal{C}$. The following proposition summarises the known relationships among the arithmetic classes,

Proposition 2 ([3, 12, 14]).

$$\mathsf{a}\text{-}\mathsf{BWrGP} = \mathsf{a}\text{-}\mathsf{AC}^0 \subseteq \mathsf{a}\text{-}\mathsf{BWBP} = \mathsf{a}\text{-}\mathsf{NC}^1 \subseteq \mathsf{a}\text{-}\mathsf{sSC}^0$$

Multilinearity and syntactic multilinearity: Multilinear and syntactic multilinear circuits are as defined in [18]. Let C be an arithmetic circuit over the ring $\mathbb{K}$, and let $X = \{x_1, \ldots, x_n\}$ be its input variables. For a gate g in C, let $P_g \in \mathbb{K}[X]$ be the polynomial represented at g. Let $X_g \subseteq X$ denote the set of variables that occur in the sub circuit rooted at g. We call C *multilinear* if for every gate $g \in C$, P_g is a multilinear polynomial. C is said to be *syntactic multilinear* if for every multiplication gate $g = h \times f$ in C, $X_h \cap X_f = \emptyset$.

In the case of formulae, the notion of multilinearity and syntactic multilinearity are (non-uniformly) equivalent.

Viewing branching programs as skew-circuits, a multilinear branching program P is one which computes multilinear polynomials at every node, and P is syntactic multilinear if in every path of the program (not just s-to-t paths), no variable appears more than once; *i.e.* the branching program is syntactic read-once.

For any arithmetic complexity class $\mathsf{a}\text{-}\mathcal{C}$, we denote by $\mathsf{ma}\text{-}\mathcal{C}$ and $\mathsf{sma}\text{-}\mathcal{C}$ respectively the functions computed by multilinear and syntactic multilinear versions of the corresponding circuits.

In [19] it is shown that the depth reduction of [23] preserves syntactic multilinearity; thus

Proposition 3 ([19]). *Any function computed by a syntactic multilinear polynomial size polynomial degree arithmetic circuit is in* sma-SAC^1.

3 Depth reduction in small width sm-circuits

This entire section is devoted to a proof of Theorem 1 below, which says that a circuit width bound can be translated to a circuit depth bound, provided the given small-width circuit is syntactic multilinear.

Theorem 1. *Let C be a* syntactic multilinear *circuit of length l and width w and circuit degree d, with $X = \{x_1, \ldots, x_n\}$ as the input variables, and constants $\{-1, 0, 1\}$ from the ring $\mathbb{K}$. Then there is an equivalent circuit C' of depth $O(w^2 \log l + \log d)$ and size $O(2^{w^2+3w} l^{25w} + 4lwd)$.*

An immediate corollary is,

Corollary 1. sma-$\mathsf{sSC}^0 \subseteq$ a-NC^1.

It can also be seen that if we apply Theorem 1 to a syntactic multilinear arithmetic circuit of poly-logarithmic width and quasi-polynomial size and degree, then we get a poly-logarithmic depth circuit of quasi-polynomial size. Thus

Corollary 2.

$$\mathsf{sma\text{-}Size}, \mathsf{Width}, \mathsf{Deg}(2^{\mathsf{poly}(\log)}, \mathsf{poly}(\log), 2^{\mathsf{poly}(\log)})$$

$$\subseteq \mathsf{a\text{-}Size}, \mathsf{Depth}(2^{\mathsf{poly}(\log)}, \mathsf{poly}(\log))$$

We first give a brief outline of the technique used. The main idea is to first cut the circuit C at length $\lceil \frac{l}{2} \rceil$, to obtain circuits A (the upper part) and B (the lower part). Let $M = \{h_1, \ldots, h_w\}$ be the gates of C at level $\lceil \frac{l}{2} \rceil$. A is obtained from C by replacing the gates in M by a set $Z = \{z_1, \ldots, z_w\}$ of new variables. Each gate g of A (or B) represents a polynomial $p_g \in \mathbb{K}[X, Z]$, and can also be viewed as a polynomial in $K[Z]$, where $K = \mathbb{K}[X]$. Since A and B are circuits of length bounded by $\lceil \frac{l}{2} \rceil$, if we can prove inductively that the coefficients of the polynomials at the output gates of A and B can be computed by small depth circuits (say $O(w \log(l/2))$, then, since p_g has at most 2^w monomials in variables from Z, we can substitute for the z_i's by the value at the output gate g_i of B (*i.e.* polynomials in $\mathbb{K}[X]$). This requires an additional depth of $O(w)$. See Figure 3.

The first difficulty in the above argument can be seen even when $w = O(1)$. Though C is syntactic multilinear, the circuit A need not be multilinear in the new dummy variables from Z. This is because there can be gates which compute large constants from $\mathbb{K}$ (*i.e.* without involving any of the variables), and hence have large degree (bounded by the degree of the circuit). This means that the polynomials in the new variables Z at the output gates of A can have non-constant degree, and the number of monomials can be large. Thus the additional depth needed to compute the monomials will be non-constant; hence the argument fails.

To overcome this difficulty, we first transform the circuit C into a new circuit C', where no gates compute "large" constants in $\mathbb{K}$. Let C be a syntactic multilinear circuit of length l

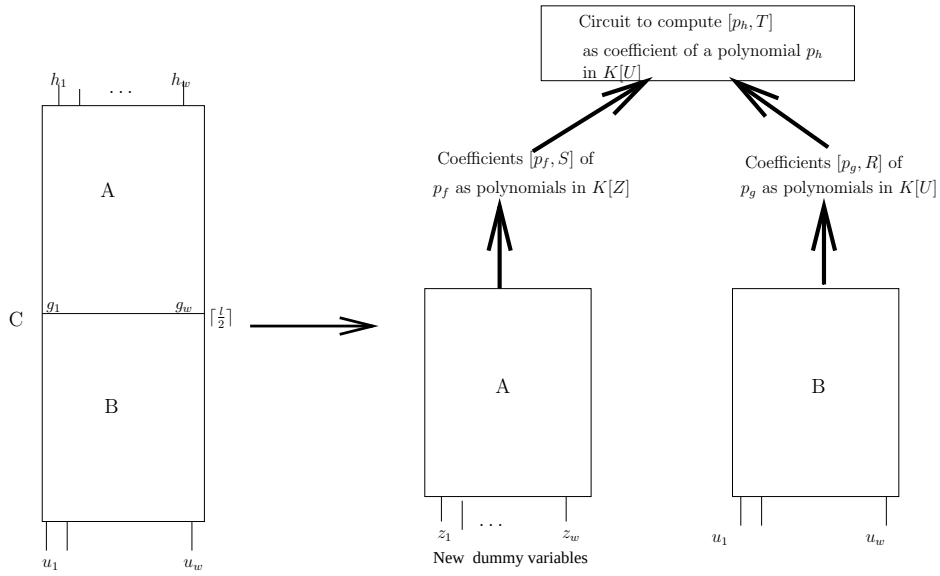

Fig. 5. Breaking up circuit C into A and B

and width w. Assume without loss of generality that every gate in C has a maximum fan-out of 2. For a gate $g \in C$, define the sets

$$\mathsf{leaf}(g) = \{h \in C \mid h \text{ is a leaf node in } C, \text{ and } g \text{ is reachable from } h \text{ in } C\}$$

$$G = \{g \in C \mid \mathsf{leaf}(g) \cap X = \emptyset\}$$

Thus G is exactly the nodes that syntactically compute constants. Now define C' as a new circuit which is the same as C except that, for all $g \in G$, we replace the i^{th} $(i = 1, 2)$ outgoing wire of g by a new variable y_{g_i}. Note that the number of such new variables introduced is at most $4lw$. (The constants can appear anywhere in the circuit. So each gate can have two new variables on its output wires and two new variables on its input wires.) Let $Y = \{y_{g_i} \mid g \in G, 1 \leq i \leq 2\}$. We show that C' is syntactic multilinear in the variables $X \cup Y$.

Lemma 1. *The circuit C' constructed above is syntactic multilinear in the variables $X \cup Y$. Further, C' does not have any constants.*

Proof. The circuit C' is clearly syntactic multilinear in the variables from X. For any gate g in C', let V_g denote $\mathsf{leaf}(g) \cap Y$. Suppose $\exists h \in C'$, $h = g \times f$, such that h is not syntactic multilinear. Then a variable y from Y must be used by f and g, so $V_f \cap V_g \neq \emptyset$. Since each variable from Y occurs on exactly one wire, this implies that there is a gate $e \in C$ such that e has a path to both f and g (e could be the head of the wire carrying y), and $y \in V_e$. Choose the highest such e (closest to h); then the $e - f$ and $e - g$ paths are disjoint. Since C is syntactic multilinear, it must be the case that $e \in G$. But by the construction above, we have $y_{e_1} \in V_f$ and $y_{e_2} \in V_g$, and due to these new variables y_{e_1} and y_{e_2}, y is not in V_f and V_g at all. (In fact, none of the variables in V_e are in V_f or V_g.) □

We now show, in Lemma 2, how to achieve depth reduction for syntactic multilinear bounded width circuits which have no constants. Then we complete the proof of Theorem 1 by explicitly computing the constants (*i.e.* the actual values represented by variables in Y) computed by the circuit C.

Lemma 2. *Let C be a width w, length l syntactic multilinear arithmetic circuit with leaves labelled from $X \cup Y$ (no constants). Then there is an equivalent arithmetic circuit D of size $O(2^{w^2+3w} l^{25w})$ and depth $O(w^2 \log l)$ which computes the same function as C.*

To establish lemma 2, we use the intuitive idea sketched in the beginning of the section; namely, slice the circuit horizontally, introduce dummy variables along the slice, and proceed inductively on each part.

Now the top part has three types of variables: circuit inputs X, variables representing constants Y as introduced in Lemma 1, and variables along the slice Z. The variables Z appear only at the lowest level of this circuit. Note that this circuit for the top part is syntactic multilinear in Z as well (because there are no constants at the leaves).

To complete an inductive proof for Lemma 2, we need to show depth-reduction for such circuits. We use Lemma 3 below, which tells us that viewing each gate as computing a polynomial in Z, with coefficients from $K = \mathbb{K}[X, Y]$, there are small-depth circuits representing each of the coefficients. We then combine these circuits to evaluate the original circuit.

More formally, let C be a width w, length l, syntactic multilinear circuit, with all leaves labelled from $X \cup Y \cup Z$ (no constants), where variables from $Z = \{z_1, \ldots z_w\}$ appear only at the lowest level of the circuit. Let $h_1, \ldots, h_w$ be the set of output gates of C *i.e.* gates at level l. Let $p_{h_i} \in \mathbb{K}[X, Y, Z]$ denote the multilinear polynomial computed at h_i. Note that p_{h_i} can also be viewed as a polynomial in $K[Z]$, *i.e.* a multilinear polynomial with variables from Z and polynomials from $\mathbb{K}[X, Y]$ as its coefficients; we use this viewpoint below. For $T \subseteq \{1, \ldots, w\}$, let $[p_{h_i}, T] \in \mathbb{K}[X, Y]$ denote the coefficient of the monomial $m_T = \prod_{j \in T} z_j$ in p_{h_i}. The following lemma tells us how to evaluate these coefficients $[p_{h_i}, T]$.

Lemma 3. *With circuit C as above, $\forall h \in \{h_1, \ldots, h_w\}$ and $T \subseteq \{1, \ldots, w\}$, there is a bounded fan-in arithmetic circuit $C^{h,T}$ of size bounded by $2^{w^2+2w} l^{25w}$ and depth $O(w^2 \log l)$, with leaves labelled from $X \cup Y \cup \{-1, 0, 1\}$, such that the value computed at its output gate is exactly the coefficient $[p_h, T]$ evaluated at the input setting to $X \cup Y$.*

Proof. We proceed by induction on the length l of the circuit.

Basis : $l = 1$. The different possibilities are as follows. Here, a is an element of $\mathbb{K}[X, Y]$.

$h = z_i z_j$: $[p_h, T] = 1$ for $T = \{i, j\}$ and 0 otherwise.
$h = a z_i$: $[p_h, T] = a$ for $T = \{i\}$ and 0 otherwise.
$h = a$: $[p_h, T] = a$ for $T = \emptyset$ and 0 otherwise.
$h = z_i + z_j$: $[p_h, T] = 1$ for $T = \{i\}$ or $T = \{j\}$ and 0 otherwise.
$h = a + z_i$: $[p_h, \emptyset] = a$, $[p_h, \{i\}] = 1$, and $[p_h, T] = 0$ otherwise.

Hypothesis: Assume that the lemma holds for all circuits D of length $l' < l$ and width w.

Induction Step: Let C be a circuit of length l, syntactic multilinear in $X \cup Y \cup Z$, where variables from Z appear only at the input level and C satisfies the conditions as in Lemma 1. Let $\{h_1, \ldots, h_w\}$ be the output gates of C. Let $\{g_1, \ldots, g_w\}$ be the gates of C at level $l' = \lceil \frac{l}{2} \rceil$.

Denote by A the circuit resulting from replacing gates g_i with new variables z'_i for $1 \leq i \leq w$, and removing all the gates below level l', and denote by B the circuit with $\{g_1, \ldots, g_w\}$ as output gates, *i.e.* gates above the g_i's are removed. We rename the output gates of A as $\{f_1, \ldots, f_w\}$. Both A and B are syntactic multilinear circuits of length bounded by l' and width w, and of a form where the inductive hypothesis is applicable. For $i \in \{1, \ldots, w\}$, p_{f_i} is a polynomial in $K[Z']$ and p_{g_i} is a polynomial in $K[Z]$, where $K = \mathbb{K}[X, Y]$.

Applying induction on A and B, for all $S, Q \subseteq \{1, \ldots, w\}$, $[p_{f_i}, S]$ and $[p_{g_i}, Q]$ have circuits $A^{f_i,S}$ and $B^{g_i,R}$. Note that $p_{h_i}(Z) = p_{fi}(p_{g_1}(Z), \ldots, p_{g_w}(Z))$. But due to multilinearity,

$$p_{f_i}(Z') = \sum_{S \subseteq [w]} \left([p_{f_i}, S] \prod_{j \in S} z'_j \right) \qquad p_{g_j}(Z) = \sum_{Q \subseteq [w]} \left([p_{g_j}, Q] \prod_{s \in Q} z_s \right)$$

Using this expression for p_{f_i} in the formulation for p_{h_i}, we have

$$p_{h_i}(Z) = \sum_{S \subseteq [w]} \left([p_{f_i}, S] \prod_{j \in S} p_{g_j}(Z) \right)$$

Hence, we can extract coefficients of p_{h_i} as follows. The coefficient of the monomial m_T, for any $T \subseteq [w]$ in p_{h_i} is given by

$$[p_{h_i}, T] = \sum_{S \subseteq [w]} [p_{f_i}, S] \left(\text{ coefficient of } m_T \text{ in } \textstyle\prod_{j \in S} p_{g_j}(Z) \right)$$

If S has t elements, then the monomial m_T is built up in t disjoint parts (not necessarily non-empty), where the kth part is contributed by the kth polynomial p_g in the above expression. So the coefficient of m_T is the product of the corresponding coefficients. Hence

$$[p_{h_i}, T] = \sum_{S = \{j_i, \ldots, j_t\} \subseteq [w]} \left([p_{f_i}, S] \sum_{\substack{Q_1, \ldots, Q_t : \\ \text{partition of } T}} \prod_{k=1}^{t} [p_{g_{j_k}}, Q_k] \right)$$

We use this expression to compute $[p_{h_i}, T]$. We first compute $[p_{f_i}, S]$ and $[p_{g_j}, Q]$ for all $i, j \in [w]$ and all $S, Q \subseteq [w]$ using the inductively constructed sub circuits. Then a circuit on top of these does the required combination. Since the number of partitions of T is bounded by w^w, while the number of sets S is 2^w, this additional circuitry has size at most $w^2 2^w w^w \leq 2^{w^2}$ (for $w \geq 2$) and depth $w \log w + w + \log w = O(w^2)$.

Let $s(l, w)$ and $d(l, w)$ denote the size and depth of the new circuit $C^{p_h,T}$. Then from the construction above, we have the recurrences

$$s(l, w) \leq 2w2^w s(l', w) + 2^{w^2} \leq 2^{2w} s(\lceil l/2 \rceil, w) + 2^{w^2}$$

$$d(l, w) \leq d(\lceil l/2 \rceil, w) + O(w^2)$$

Note that $l' = \lceil l/2 \rceil$ satisfies $l' \leq 3l/4$. By induction, $s(l', w) \leq 2^{w^2+2w}(l')^{cw}$ for some constant c to be chosen later. So

$$\begin{aligned} s(l,w) &\leq 2^{2w}2^{w^2+2w}(l')^{cw} + 2^{w^2} &&\leq 2^{w^2}2^{4w}(3l/4)^{cw} + 2^{w^2} \\ &\leq 2^{w^2}\left[2^{4w}(3l/4)^{cw} + 1\right] &&\leq 2^{w^2}\left[2^{4w}(3l/4)^{cw}\right]2 \\ &= 2^{w^2+2w}l^{cw}\left[2^{2w}(3/4)^{cw}2\right] \leq 2^{w^2+2w}l^{cw} \end{aligned}$$

where the last inequality holds whenever $8(3/4)^c \leq 1$, say $c \geq 25$.

Similarly, solving the recurrence for $d(l, w)$ gives $d(l, w) = O(w^2 \log l)$. □

Now, finally, we can establish Lemma 2.

Proof (of lemma 2). We first relabel all the nodes at the lowest level by new variables $z_1, \ldots, z_w$. Then, applying Lemma 3, we obtain circuits for $[p_g, T]$, where g is an output gate of C and $T \subseteq \{1, \ldots, w\}$. Now, to compute p_g, we sum over all T the values $[p_g, T] \times \prod_{j \in T} val(z_j)$, where $val(z_j)$ denotes the original variable for which z_j was substituted. This adds $O(w)$ to the overall depth of the circuit, thus resulting an overall depth of $O((w + w^2 \log l)) = O(w^2 \log l)$. The resulting circuit size is bounded by $O(s2^w)$, where s is an upper bound on the size of the circuits constructed in Lemma 3, and hence is bounded by $O(2^{w^2+3w}l^{25w})$ □

And with lemma 2 established, we can now get the desired depth-reduction result.

Proof (of Theorem 1). Given circuit C, we construct C' as per Lemma 1, and then apply Lemma 2 to obtain an equivalent circuit D of depth $O(w^2 \log l)$ and size $O(2^{w^2+2w}l^{25w})$, which uses variables from $X \cup Y$. To eliminate variables from Y, let $val(y_{g_i})$ denote the value of the gate g in the original circuit C. Since the degree of C is d and C uses only the constants $\{-1, 0, 1\}$, we have $-2^d \leq val(y_{g_i}) \leq 2^d, \forall g \in G, i \in \{1, 2\}$. These values require at most $d + 1$ bits for their representation. For every $r \in \{-2^d, \ldots, 0, \ldots, 2^d\}$, there is an arithmetic circuit of depth $O(\log d)$ and size $O(d)$ which computes r. Now replace all occurrences of variables $y_{g_i} \in Y$ by the (hardwired) arithmetic circuit which computes $val(y_{g_i})$. The new circuit D' thus constructed has size $O(2^{w^2+3w}l^{25w} + 4lwd)$ and depth $O(w^2 \log l + \log d)$. □

Note that the last step in the construction above hardwires constants explicitly computed by C. In this sense (and only because of this), the overall construction is not uniform.

Remark 1. If the constant-width circuit C we start with is multilinear but not syntactic multilinear, then the circuit C' as in Lemma 1 need not be multilinear in the variables $X \cup Y$. This is one place where the above construction crucially uses syntactic multilinearity, and does not generalise to multilinear circuits. See Figure 6 for an example.

Remark 2. If the circuit is allowed to use arbitrary constants from $\mathbb{K}$, then Lemma 1 is not needed in the above construction.

Even so, the result does not generalise to multilinear C, because Lemma 3 requires syntactic multilinearity in the slice variables Z.

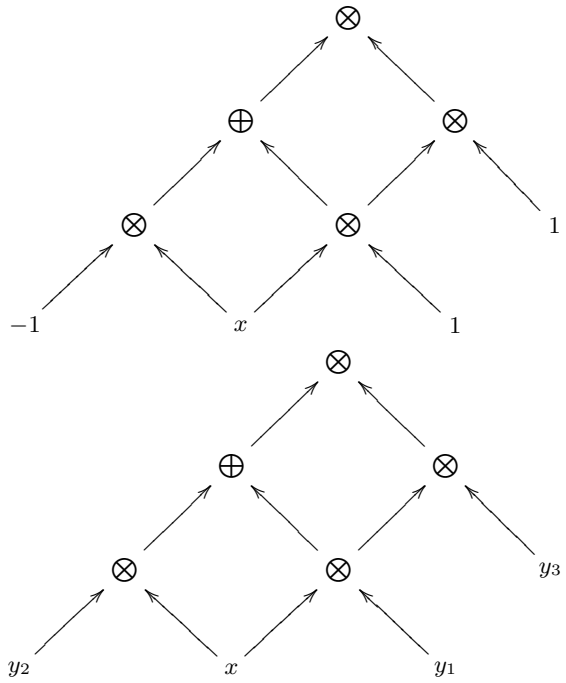

Fig. 6. C' is not multilinear after reduction as in Lemma 1

4 Relationships among syntactic multilinear classes

This section explores the relationships among the syntactic multilinear versions of the arithmetic classes which are related to NC^1.

A classical result from [9] shows that for every arithmetic formula F of size s, there is an equivalent arithmetic formula F' which has depth $O(\log s)$ and size $\mathsf{poly}(s)$. A careful observation of this proof shows that if we start with a syntactic multilinear formula F, then the depth-reduced formula F' is also syntactic multilinear.

Theorem 2. *Every syntactic multilinear formula with n leaves has an equivalent syntactic multilinear circuit of depth $O(\log n)$ and size $O(n)$.*
In particular, $\mathsf{sma\text{-}F} \subseteq \mathsf{sma\text{-}NC}^1$.

Proof. By simultaneous induction on the number of leaves in the formula, we can prove the following statements. This is exactly the construction of [9], analysed carefully for syntactic multilinearity.

(i) If F is a syntactic multilinear formula with n leaves, then there is an equivalent syntactic multilinear circuit F' of depth $\lceil 4 \log n \rceil$ and size $2n$.
(ii) If x is any leaf in F, then we can express F as $F' = Ax + B$, where A, B are syntactic multilinear, do not depend on x and and are of depth $\lceil 4 \log n \rceil$.

In the base case, there is either a single variable or a constant, and the claim holds trivially.

Let X be a tree separator for F, with children L, R, so that $X = L \mathsf{op} R$. Replace the whole subtree under X by a new variable x. By inductive statement (ii), we have $F" = Ax+B$ where A, B are as above (*i.e.* they are both syntactic multilinear and do not depend on X). Also by inductive statement (i), we have syntactic multilinear formula L', R' equivalent to L, R of small depth. Thus we have $F' = A.(L' \mathsf{op} R') + B$. Since A does not depend on any variable below X, F' is syntactic multilinear. Also we can see that it has the required depth.

To prove the second half of the statement above, let x be any leaf in F. Now find a tree separator $X = L \mathsf{op} R$ such that the subtree at one of its children, say L, contains x as a leaf and is of size $< n/2$. Then, by inductive statement (ii) applied to L, $L' = Ax + B$, where A, B are independent of x, syntactic multilinear and of small depth. Now replace the subtree at X by a new variable y. Applying inductive statement (ii), we have $F' = Cy + D$, where C, D are syntactic multilinear small depth formulae which do not depend on y (*i.e.* $L \mathsf{op} R$). Applying inductive statement (i) to R, we have an equivalent small-depth R'.

Case 1: $\mathsf{op} = +$. Then $F' = C((Ax + B) + R') + D = CAx + (CB + CR' + D)$. This is again syntactic multilinear since C does not depend on y, *i.e.* $Ax + B + R$.

Case 2: $\mathsf{op} = \times$. Then $F' = C(Ax + B)R' + D = CAR'x + (CBR' + D)$. Here again F' is syntactic multilinear since C does not depend on A, B, R', and also because A and B do not share any variables with R'.

Since we are constructing a circuit and not a formula, we don't need to replicate the circuits for C and R'. For details about the size/depth, see the analysis in [9]. □

It is easy to see that the path-preserving simulation of a constant width branching program by a log depth circuit preserves syntactic multilinearity:

Lemma 4. *For any syntactic multilinear branching program P of width w and size s over ring $\mathbb{K}$, there is an equivalent syntactic multilinear circuit C of depth $O(\log s)$ and size $O(s)$ with fan-in of $+$ gate bounded by w (or alternatively, depth $O(\log w \log s)$ and bounded fan-in).*
In particular, sma-BWBP $\subseteq$ sma-NC^1 *and* sma-BP $\subseteq$ sma-SAC^1.

Proof. Let l be the length of P ($s = lw$), and let $p_{s,t}$ denote the weighted sum of the directed paths between nodes s and t. Let $v_1, \ldots v_w$ denote the nodes at the level $l' = \lceil l/2 \rceil$ of P. Then $p_{s,t} = \sum_{i=1}^{w} p_{s,v_i} \times p_{v_i,t}$. Thus the depth and size of the inductively constructed circuit satisfy the recurrences $d(l) = 2 + d(l')$ and $s(l) = (3w)s(l')$, giving the desired bounds. It is clear that the circuit so constructed is syntactic multilinear; if it weren't, the offending $\times$ gate would pinpoint a path in P that reads some variable twice. □

It is also straightforward to see that the construction of [13], staggering a small-depth formula into a small-width one, preserves syntactic multilinearity. Thus

Lemma 5. *Let Φ be any sm-formula with depth d and size s. Then there is an equivalent syntactic multilinear formula Φ' of length $2s$ and width d.*
In particular, sma-NC^1 $\subseteq$ sma-LWF.

Proof. For completeness we give a detailed proof here. The construction is by induction on the structure of the formula Φ. The base case is when Φ is a single variable or a constant, in which case the lemma holds trivially.

Suppose the lemma holds for any formula of depth at most $d-1$. Consider the root gate f of a formula Φ of depth d. Suppose $f = \sum_{i=1}^{k} g_i$ (respectively $f = \prod_{i=1}^{k} g_i$). As the depth of each formula g_i is bounded by $d-1$, by induction we have formulae g_i' of width $d-1$ and length bounded by s_i (the size of g_i), computing the same function as g_is. Place the node corresponding to f with two children. At one child, place the formula g_1'; at the other, place a series of no-op (*i.e.* $\times 1$ or $+0$) gates till the last level of g_1'. Then give the last no-op gate two children, place g_2' at one child, and so on. The width of the new formula Φ' thus obtained is bounded by $\max_i \mathsf{width}(g_i') + 1$, and its length is bounded by $\sum_i \mathsf{length}(g_i') + 1 \leq \sum_i s_i + 1 \leq s$. Note that in this process, for any gate g in Φ the variables it operates on are not changed in the new formula Φ', that is, the only new gates which are introduced in Φ' are the no-op gates which are used for staggering, which only multiply by the constant 1. Thus if Φ is syntactic multilinear then so is Φ. □

From Lemma 5 and Theorem 2, we have the following equivalence.

Corollary 3. *Over any ring* $\mathbb{K}$,
sma-F= sma-LWF= sma-NC^1= sma-Formula-Depth,Size(log, poly).

A straightforward inductive construction of a branching program from a log depth formula results in a log width BP and preserves syntactic multilinearity. But the reverse containment may not hold. However, by restricting the branching program as in [3], we can obtain a characterisation for a-NC^1 which also preserves syntactic multilinearity. In [3] a characterisation for bounded depth arithmetic circuits in terms of counting number of paths in a restricted version of bounded width grid graphs is presented. We note that the characterisation given in [3] works for bounded depth arithmetic circuits over arbitrary rings, showing that a-BWrGP = a-AC^0. By closely examining the parameters in [3], we obtain a characterisation for a-NC^1 in terms of the restricted version of log width grid branching programs. We also note that these constructions preserve syntactic multilinearity. In the statement and proof below, we use the notion of alternation-depth: a circuit C has alternation depth a if on every root-to-leaf path, the number of maximal segments of gates of the same type is at most a. Also, for an rGP (and in fact any branching program) P, we denote by $\mathsf{Var}(P)$ the set of variables that appear on some s-to-t path in P. For a formula F, $\mathsf{Var}(F)$ denotes the variables appearing anywhere in the formula F; if h is the root of F, then without loss of generality $\mathsf{Var}(F) = X_h$.

Lemma 6. *Let Φ be an arithmetic formula of size s (i.e. number of wires) and alternation-depth $2d$ over $\mathbb{K}$ and with input variables $X \in \mathbb{K}^n$. Then there is a restricted grid program P of length $s^2 + 2s$ (i.e. the number of edge layers) and width $\max\{2, 2d\}$, where the edges are labelled from $\mathsf{Var}(\Phi) \cup \mathbb{K}$, such that the weighted sum of s-to-t paths in P is equal to the function computed by Φ.*
Further, if Φ is syntactic multilinear, then so is P.

Proof. The construction here is exactly the same as in [3]; it is included here for completeness in arguing, over more general parameters, that syntactic multilinearity is preserved. Without loss of generality, assume that the formula Φ is such that all nodes in a particular layer represent the same type of gate and two successive layers have different kind of gates. Also, assume that Φ is height balanced, *i.e.* any root to leaf path in Φ is of length exactly $2d$. Further assume that the root is a $\times$ gate. If these conditions do not hold, then ensuring them will blow up the size of Φ to at most s^2, and increase the depth by at most 2. We assume that s and a are the size and alternation depth of a formula already in this normal form.

We proceed by induction on the depth of the formula Φ. The base case is when $d \leq 1$. If the depth is 0, then Φ is either a variable or a constant in the underlying ring. In this case the graph is $G_{0,1}(c)$ where $\Phi = c$. If $d = 1$, then Φ is a product of linear factors, and a suitable composition of $G_{0,1}(c)$ graphs and $G_{0,2}$ represents it.

Suppose that for any (syntactic multilinear) formula F with alternation depth $2d' < 2d$ and size s' (in the normal form described above), there is a (syntactic multilinear) restricted grid program P of width $2d'$ and length $s'^2 + 2s'$, where P uses variables from $\mathsf{Var}(F)$.

Now let Φ be a normal form formula with alternation depth $2d$. Consider the root gate g of Φ. Let $g_1, \ldots, g_k$ be the children of g, where $g_i = \sum_{j=1}^{t_i} g_{i_j}$. Let s_{i_j} and $2d_{i_j} = 2d - 2$ respectively denote the size and alternation depth of the sub formula rooted at g_{i_j}. Note that $s = k + \sum_i (t_i + \sum_j s_{i_j})$. Applying induction on the sub-formula rooted at each g_{i_j}, let Q_{i_j} denote the resulting restricted grid program for the formula at g_{i_j}. Now place the Q'_{i_j}s $(1 \leq j \leq t_i)$ as in Figure 8 to get the program P_i, and connect the P_i's as shown in Figure 7 to get the desired program P. By the inductive hypothesis, $\mathsf{length}(Q_{i_j}) \leq s_{i_j}^2 + 2s_{i_j}$ and $\mathsf{width}(Q_{i_j}) \leq 2d_{i_j}$. From the construction as above, we have $\mathsf{length}(P_i) = t_i + 1 + \sum_j \mathsf{length}(Q_{i_j}) \leq t_i + 1 + \sum_j (s_{i_j}^2 + 2s_{i_j})$ and hence $\mathsf{length}(P) = k - 1 + \sum_i \mathsf{length}(P_i) \leq k - 1 + \sum_i ((t_i + 1) + \sum_j (s_{i_j}^2 + 2s_{i_j})) \leq s^2 + 2s$. Note that the construction in Figure 8 adds 2 to the width and the construction in Figure 7 does not change the width. Hence the width of P is bounded by $2 \max_{i,j} d_{i_j} + 2 = 2d$.

If Φ is syntactic multilinear, then the formulae rooted at g_{i_j} are all syntactic multilinear, and for $i \neq i'$, $\mathsf{Var}(g_i) \cap \mathsf{Var}(g_{i'}) = \emptyset$. Thus, by the inductive hypothesis, the programs Q_{i_j} are syntactic multilinear, and only use variables from $\mathsf{Var}(g_{i_j})$, and hence the programs P_i (for each i) only use variables from $\mathsf{Var}(g_i)$. Thus for every $i \neq i'$, $\mathsf{Var}(P_i) \cap \mathsf{Var}(P_{i'}) = \emptyset$. Since each path in the final program goes through exactly one Q_{i_j} for each i, it follows that P is syntactic read-once. □

We now establish the converse to Lemma 6. The proof of the converse as in [3] is uniform and it produces a circuit rather than a formula. If we do not insist on uniformity of the circuit, then we actually get a formula. Thus it can be shown that functions computed by width $2w+2$, length l restricted grid programs can be computed (non uniformly) by formulae of depth $2w + 2$ and size $O(l)$.

Lemma 7. *Let P be an arithmetic rGP of length l (number of edge layers) and of width $2w + 2$ with variables from $X \in \mathbb{K}$. Then there exists an equivalent arithmetic formula Φ over $\mathbb{K}$, with alternation depth at most $2w + 2$, size (number of wires) at most $2l$, and*

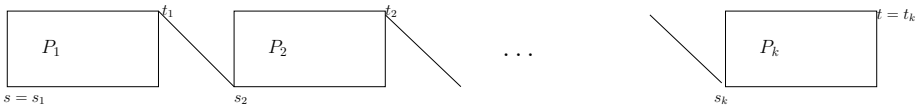

Fig. 7. Multiplication of rGP's

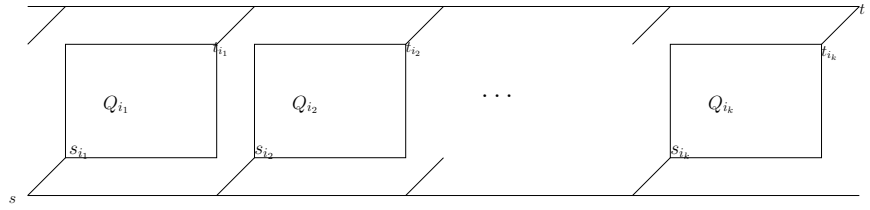

Fig. 8. Addition of rGP's

$\mathsf{Var}(\Phi) = \mathsf{Var}(P)$.
Further, if P is syntactic multilinear, then so is Φ.

Proof. Again, this construction the same as in [3]; it is presented here with the induction unfolded to allow arguing, over more general parameters, that syntactic multilinearity is preserved.

For a program B, let $f(B)$ denote the the function computed by B. We proceed by induction on w. The base case is when $w = 0$, *i.e.* we have a rGP P of width 2. Then $f(P)$ can be computed by a depth 2 circuit with one $\times$ gate as root and a number of $+$ gates as its inputs, where the $+$ gates get input from $X \cup \mathbb{K}$. The total fan-in of the $+$ gates is bounded by the number of layers which contain the graph $G_{0,1}(c)$, for some c. The fan-in of the $\times$ gate is one more than the number of layers which have the graph $G_{0,2}$. (The layers having $G_{0,0}$ do not contribute to the formula.) Thus the total number of wires is bounded by $l + 1 \leq 2l$, and depth is 2. If P is syntactic multilinear, no path reads the same variable twice, and so the blocks separated by $G_{0,2}$ have disjoint sets of variables. Hence the top $\times$ gate operates on disjoint sets of variables.

Suppose that for any $w' < w$ the claim holds, *i.e.* for a (syntactic multilinear) rGP P' of width $2w' + 2$ and length l', there is an equivalent (syntactic multilinear) formula Φ' of depth $2w' + 2$ and size $2l'$ and using only variables from $\mathsf{Var}(P')$.

Now P is the given rGP of width $2w + 2$, length l. Let P be composed as $g_1, \ldots, g_l$. Let $i_1 < i_2 < \ldots < i_m$ be the (uniquely defined) set of all indices where $g_{i_1}, \ldots, g_{i_m}$ are the graph $G_{w,2w+2}$. Define $i_0 = 0$, $i_{m+1} = l + 1$.

For each $0 \leq j \leq m$, let P_j denote the program $g_{i_j+1}, \ldots, g_{i_{j+1}-1}$ sandwiched between the jth and $(j + 1)$th incidence of $G_{w,2w+2}$. The nodes s_j and t_j for each P_j are defined accordingly. Let l_j denote the length of P_j; then $l = m + \sum l_j$. Note that these P_js do not have $G_{w,2w+2}$ at any layer, and $f(P) = \prod_j f(P_j)$.

Consider P_j for some j. Let $h_{j_1}, \ldots h_{j_{r_j}}$ denote the layers of P_j which are the connecting graph $G_{w,2w+1}$. Let $Q_{j,k}$ denote the part of the program between h_{j_k} and $h_{j_{k+1}}$, and $Q_{j,0}$ denote the part between g_{i_j} and h_{j_1} and Q_{j,r_j} denote the part between h_{j_r} and $g_{i_{j+1}}$. Let $Q'_{j,k}$

denote the graph obtained from $Q_{j,k}$ be removing the top-most and bottom-most lines and the edges connecting them. Then $\mathsf{width}(Q'_{j,k}) = \mathsf{width}(Q_{j,k}) - 2 = 2w$. Let $l_{j,k}$ denote the length of $Q'_{j,k}$; so $l_j \leq r_j + \sum_{k=1}^{r_j-1} l_{l,k}$. The nodes $s'_{j,k}$ and $t'_{j,k}$ for $Q'_{j,k}$ are defined accordingly. Now $f(P_j) = \sum_{k=1}^{r_j-1} f(Q'_{j,k})$. (Note that $Q_{j,0}$ and Q_{j,r_j}, even if non-trivial, play no role in $f(P_j)$ because there is no connection from s_j to these blocks.)

By induction, for each $Q'_{j,k}$ we obtain equivalent (syntactic multilinear) formula $\Phi_{j,k}$ with variables from $\mathsf{Var}(Q'_{j,k})$, $\mathsf{size}(\Phi_{j,k}) = s_{j,k} = 2l_{j,k}$ and $\mathsf{depth}(\Phi_{j,k}) = d_{j,k} = 2w$. Now define $\Phi = \prod_j \sum_{k=1}^{r_j-1} \Phi_{j,k}$. Then $\mathsf{size}(\Phi) = s = m + \sum_j (r_j - 1 + \sum_k 2l_{j,k}) \leq 2l$ and $\mathsf{depth}(\Phi) = 2w+2$ as desired. Clearly $\mathsf{Var}(\Phi) = \mathsf{Var}(P)$.

If P is syntactic multilinear, then inductively we have $\Phi_j = \sum_{k=1}^{r_j} \Phi_{j,k}$ operating on $\mathsf{Var}(P_j)$, and each $\Phi_{j,k}$ is syntactic multilinear. Consider the root gate of Φ. If it is not syntactic multilinear, then for some $j < j'$, and for some k, k', $\Phi_{j,k}$ and $\Phi_{j',k'}$ use the same variable x. Thus, by induction, P_j has an s_j-to-t_j path using x, and $P_{j'}$ also has an $s_{j'}$-to-$t_{j'}$ path using x. Combining these paths with (1) the s-to-s_j path along the bottom-line, (2) the t_j-to-$s_{j'}$ path using g_{i_j+1} and then the bottom line, and (3) the $t_{j'}$-to-t path along the top line, gives a path in P that reads x twice, contradicting the read-once property of P. □

As an immediate consequence of the above two lemmas, we have:

Corollary 4. *1.* $\mathsf{sma\text{-}AC^0} = \mathsf{sma\text{-}BWrGP}$
2. $\mathsf{sma\text{-}NC^1} = \mathsf{sma\text{-}LWrGP}$*;*
3. $\mathsf{a\text{-}NC^1} = \mathsf{a\text{-}LWrGP}$

Note that the above construction also holds in the case of boolean circuits. Hence we have the following characterisation for $\mathsf{NC^1}$.

Corollary 5. $\mathsf{NC^1} = \mathsf{LWrGP}$.

Thus we get a characterisation for $\mathsf{NC^1}$ and $\mathsf{a\text{-}NC^1}$ in terms of a restricted class of log width polynomial size planar branching programs.

In [5] it is shown that any $O(\log n)$ depth polynomial size formula has an equivalent 3-register straight line program. This proves that any arithmetic $\mathsf{NC^1}$ circuit has polynomial size constant width arithmetic branching programs *i.e.* $\mathsf{a\text{-}NC^1} \subseteq \mathsf{a\text{-}BWBP}$. Can the same be stated for $\mathsf{sma\text{-}NC^1}$ and $\mathsf{sma\text{-}BWBP}$? That is, if the given formula is syntactic multilinear, then does the resulting branching program have the syntactic multilinear property? It turns out that this is not the case. In fact the resulting program need not even be multilinear. Figure 9 illustrates a simple example where the formula is syntactic multilinear, but the resulting branching program is not even multilinear.

From the example in Figure 9, it can be seen that any nested multiplication in the formula leads to violation of multilinearity at some of the nodes (node u in Figure 9 computes $p(u) = xy^2z$).

5 Skew formulae

In this section, we consider a question motivated by the setting of Valiant's algebraic complexity classes defined in [22]. VP is the class of polynomials of polynomial degree, computable

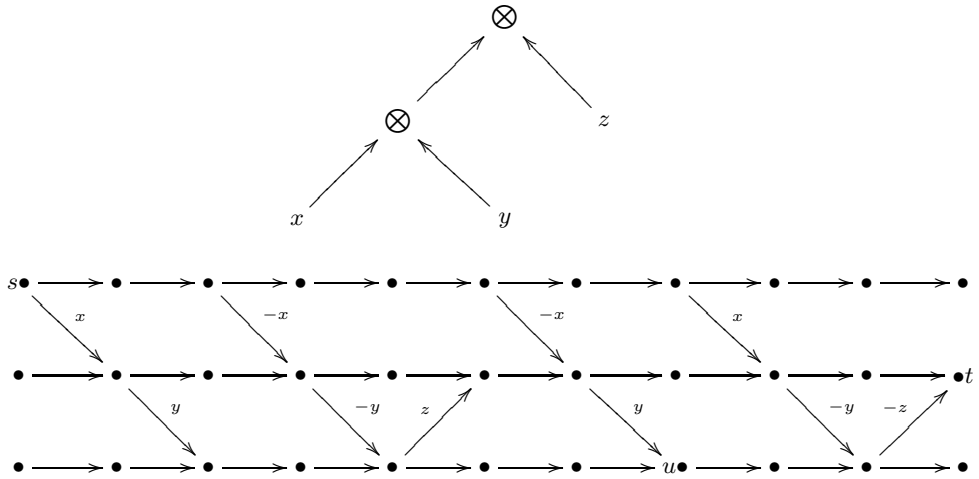

Fig. 9. An example where the Ben-Or and Cleve construction does not preserve multilinearity as $p(u) = xy^2z$

by polynomial-sized circuits. Similarly one can define VF, VNC^1, and so on. VNP is the class of polynomials expressible as

$$p(x_1, \ldots, x_n) = \sum_{e \in \{0,1\}^m} g(X, e)$$

where $m \in O(\mathsf{poly}(n))$ and the polynomial g is in VP. Thus, loosely speaking, VNP equals $\sum \cdot \mathsf{VP}$. See [10, 11] for more details; see also [15, 16].

It is well known that the complexity class NP is equivalent to $\exists \cdot \mathsf{P}$ and in fact even to $\exists \cdot \mathsf{F}$. A similar result holds in the case of Valiant's algebraic complexity classes too. Valiant has shown that $\mathsf{VNP} = \sum \cdot \mathsf{VF}$, and thus the polynomial g in the expression above can be assumed to be computable by a formula of polynomial size and polynomial degree.

Noting that VNP is the class of polynomials which are projection equivalent to the "permanent" polynomial, a natural question arises about the polynomials which are equivalent to determinant. Since the determinant exactly characterises the class of polynomials which are computable by skew arithmetic circuits ([21]), the question one could ask is: can the determinant be written as an exponential sum of partial instantiations of a polynomial that can be computed by *skew formula* of poly size, SkewF? Recall that a circuit is said to be skew if every $\times$ (or $\wedge$) gate has at most one child that is not a circuit input. Skew circuits are essentially equivalent to branching programs. Thus one could ask the related question: since $\mathsf{VP} \subseteq \sum \cdot \mathsf{VP} = \sum \cdot \mathsf{VF}$, can we show that $\mathsf{VSkew} \subseteq \sum \cdot \mathsf{VSkewF}$?

We show that this is highly unlikely. We first give an equivalent characterisation of $\mathsf{a\text{-}SkewF}$ (Lemma 8) placing it inside $\mathsf{a\text{-}AC}^0$ (see [1, 2]), and then use it to show that $\sum \cdot \mathsf{VSkewF}$ is in fact contained in VNC^1 (Theorem 3). Thus placing VSkew in $\sum \cdot \mathsf{VSkewF}$ is analogous to the statement that GapL equals GapNC^1, which we believe is quite unlikely.

5.1 A characterisation of VSkewF

Lemma 8. *Let $f \in \mathbb{Z}[X]$ be a polynomial computed by a skew formula (with constants from $\{-1, 0, 1\}$) of size s. Then the degree of f, the number of non-zero monomials in f, and the absolute value of each coefficient are all bounded by s.*
Conversely, let $f \in \mathbb{Z}[X]$ be a polynomial of degree d, where m monomials have non-zero coefficients and the absolute value of each coefficient is bounded by c. Then f can be computed by a skew formula of size $O(mdc)$ with constants from $\{-1, 0, 1\}$.

Proof. Let F be a skew formula of size s. Consider a proving subtree T of F. Since F is skew, T looks like a path, with edges hanging out at nodes labelled $\times$. But in a tree, the number of root to leaf paths is bounded by the number of leaves in the tree. Thus the number of proving subtrees of F is at most s. Let $p_F \in \mathbb{Z}[X]$ be the polynomial computed by the formula F, where X is the set of input variables of F. Since a proving subtree in F corresponds to a monomial in p_F, we have that the number of non-zero monomials in p_F is bounded by s. Since the degree of the monomial contributed by such a path is at most the length of the path, the degree of p_F is at most s.

Further, each leaf-to-root path contributes a monomial with coefficient in $\{-1, 0, 1\}$. This is because we only allow constants $-1, 0, 1$ at leaves, and the circuit is skew. Thus, the overall coefficient of a monomial is at most the number of paths, and so is bounded (in absolute value) by s.

On the other hand, if a polynomial $p \in \mathbb{Z}[X]$ has t non-zero monomials $m_1, \ldots, m_t$, and the coefficient c_i of each m_i is bounded in absolute value by c, then we can construct a skew formula which explicitly makes c_i copies of m_i for each i and adds them up. This formula computes p in size $O(mdc)$. □

Corollary 6. $\mathsf{a\text{-}SAC}^0 \subset \mathsf{a\text{-}SkewF} \subset \mathsf{a\text{-}AC}^0$.

Proof. The containments follow directly from Lemma 8. To see why they are proper: (1) Even over the Boolean setting, the function $\oplus_{i=1}^{\log n} x_i$ is in SkewF but not in SAC^0. Any Boolean function sensitive to only $O(\log n)$ of its n inputs is in SkewF. Functions computed by a $\mathsf{a\text{-}SAC}^0$ circuit have $O(1)$ degree, and so cannot equal the class of poly-degree poly-support polynomials a-SkewF. (2) The function $\prod_i (x_i + y_i)$ is in $\mathsf{a\text{-}AC}^0$ but not in a-SkewF because it has too many monomials. □

5.2 An upper bound for $\sum$.VSkewF

Theorem 3. *Let $f \in \mathbb{Z}[X]$ be expressible as $f(X) = \sum_{e \in \{0,1\}^m} \phi(X, e)$, where ϕ has a poly size* skew *formula. Then $f \in \mathsf{VNC}^1$.*
In other words, $\sum \cdot \mathsf{VSkewF} \subseteq \mathsf{VNC}^1$.

Proof. Since $\phi(X, Y)$ (where $X = X_1, \ldots, X_n$ and $Y = Y_1, \ldots, Y_m$) has a poly size skew formula, by Lemma 8 we know that the number of non-zero monomials in ϕ is bounded by some polynomial $q(n, m)$. Hence the number of non-zero monomials in $\phi(X, Y)|_X$ (*i.e.* , monomials in X with coefficients from $\mathbb{Z}[Y]$) and hence in $f(X)$, is also bounded by $q(n, m)$.

For any $\alpha \in \mathbb{N}^n$, consider the monomial $X^\alpha = \prod_{\alpha_i} X_i^{\alpha_i}$, and define the set S_α as

$$S_\alpha = \{\beta \in \{0,1\}^m \mid X^\alpha Y^\beta \text{ has a non-zero coefficient } a_{\alpha,\beta} \text{ in } \phi\}$$

Clearly, for each α, we have $|S_\alpha| \leq q(n,m)$.

Since $\phi(X,Y)$ is evaluated only at Boolean settings of Y, we can assume, without loss of generality, that it is multilinear in Y. So it can be written as

$$\phi(X,Y) = \sum_{\alpha \in \mathbb{N}^n} \sum_{\beta \in \{0,1\}^m} a_{\alpha,\beta} X^\alpha Y^\beta$$

Hence we have the following:

$$\begin{aligned} f(X) &= \sum_{e \in \{0,1\}^m} \sum_{\alpha \in \mathbb{N}^n} \sum_{\beta \in \{0,1\}^m} a_{\alpha,\beta} X^\alpha e^\beta \\ &= \sum_{\alpha \in \mathbb{N}^n} \left(X^\alpha \sum_{\beta \in S_\alpha} \left[a_{\alpha,\beta} \sum_{e \in \{0,1\}^m} e^\beta \right] \right) \\ &= \sum_{\alpha \in \mathbb{N}^n} \left(X^\alpha \sum_{\beta \in S_\alpha} a_{\alpha,\beta} 2^{m-l_\beta} \right) \end{aligned}$$

where l_β = number of 1's in the bit vector $\beta \in \{0,1\}^m$.

Then the coefficient c_α of X^α in $f(X)$ is given by $\sum_{\beta \in S_\alpha} a_{\alpha,\beta} 2^{m-l_\beta}$. Since $|S_\alpha|$ and the values $|a_{\alpha,\beta}|$ for each $\beta \in S_\alpha$ are bounded by a polynomial in $|\phi|$, c_α can be computed by a VNC^1 circuit. Since there are only a polynomially many non-zero monomials in $f(X)$, with an additional log depth we can compute $f(X)$. □

Thus, if the Determinant polynomial is expressible as $\sum.\mathsf{VSkewF}$ then it belongs to VNC^1.

5.3 Multilinear Versions

Here we consider the multilinear versions of the skew formulae. From Lemma 8, we know that $\mathsf{a\text{-}SkewF}$ is characterised by polynomials with polynomial many coefficients. The construction yields, for any multilinear polynomial computed by a skew formula, an equivalent skew formula which is syntactic multilinear. Hence the notion of multilinearity and syntactic multilinearity are the same for skew formulae.

Since any multilinear polynomial that can be computed by an $\mathsf{a\text{-}SAC}^0$ circuit has a small number of monomials, the containments of corollary 6 hold in the syntactic multilinear case too. Also, note that the polynomial $\prod_i (x_i + y_i)$ is multilinear, and can be computed by a $\mathsf{sma\text{-}AC}^0$ circuit.

Corollary 7. $\mathsf{sma\text{-}SAC}^0 \subset \mathsf{sma\text{-}SkewF} = \mathsf{ma\text{-}SkewF} \subset \mathsf{sma\text{-}AC}^0$

6 Conclusion

This work came out of an attempt to close the gap in $\mathsf{a\text{-}NC}^1 \subseteq \mathsf{a\text{-}sSC}^0$. We have not been able to do this; we can only show that $\mathsf{sma\text{-}sSC}^0 \subseteq \mathsf{a\text{-}NC}^1$. Can the depth-reduction be pushed to all of $\mathsf{a\text{-}sSC}^0$? At least $\mathsf{ma\text{-}sSC}^0$? Alternatively, can the depth-reduced circuit be made multilinear?

Another unsettled question is to understand the Boolean containments $\mathsf{NC}^1 = \mathsf{LWrGP} \subseteq \mathsf{LWGP} \subseteq \mathsf{LWBP} \subseteq \mathsf{sSC}^1 \subseteq \mathsf{SC}^1 = \mathsf{L}$. Where exactly does the power of the classes actually jump from NC^1 to L?

It would also be interesting see if the constructions described here can be made uniform.

A very interesting problem arising from [18, 19] and left open even after this study is whether the proper separation between $\mathsf{sma\text{-}NC}^1$ and $\mathsf{sma\text{-}SAC}^1$ can be improved. Note that SAC^1 and BP in the Boolean setting equal LogCFL and NL respectively; thus, Raz's result separates NC^1 from LogCFL in the syntactic multilinear setting. And LogCFL and NL are very close; all known structural results for one hold for the other too. So an obvious question is whether the proper separation can be pushed to the syntactic multilinear version of NL, namely $\mathsf{sma\text{-}BP}$. Neither of the separating functions from [18, 19] seems to be in $\mathsf{sma\text{-}BP}$. However, the interpolation between NC^1 and SAC^1 is via fan-in of $+$ gates in log-depth circuits, whereas that between NC^1 and NL is via width of branching programs. So perhaps the correct question to ask is whether $\mathsf{sma\text{-}BWBP}$ (which is contained in, but not yet known to equal, $\mathsf{sma\text{-}NC}^1$) is separate from $\mathsf{sma\text{-}BP}$. It would be very interesting to show such a separation, and it also seems accessible with current techniques, given the wealth of results about syntactic read-once branching programs.

References

1. M. Agrawal, E. Allender, and S. Datta. On TC^0, AC^0, and arithmetic circuits. *Journal of Computer and System Sciences*, 60(2):395–421, 2000.
2. E. Allender. Arithmetic circuits and counting complexity classes. In J. Krajicek, editor, *Complexity of Computations and Proofs*, Quaderni di Matematica Vol. 13, pages 33–72. Seconda Universita di Napoli, 2004. An earlier version appeared in the Complexity Theory Column, SIGACT News 28, 4 (Dec. 1997) pp. 2-15.
3. E. Allender, A. Ambainis, D. A. Barrington, S. Datta, and H. LêThanh. Bounded depth arithmetic circuits: Counting and closure. In *International Colloquium on Automata, Languages, and Programming ICALP*, ICALP'99, pages 149–158, 1999.
4. D. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38(1):150–164, 1989.
5. M. Ben-Or and R. Cleve. Computing algebraic formulas using a constant number of registers. *SIAM J. Comput.*, 21(1):54–58, 1992.
6. B. Bollig, S. Waack, and P. Woelfel. Parity graph-driven read-once branching programs and an exponential lower bound for integer multiplication. *Theor. Comput. Sci.*, 362(1-3):86–99, 2006.
7. B. Bollig and P. Woelfel. A lower bound technique for nondeterministic graph-driven read-once-branching programs and its applications. *Theory Comput. Syst.*, 38(6):671–685, 2005.
8. A. Borodin, A. A. Razborov, and R. Smolensky. On lower bounds for read-k-times branching programs. *Computational Complexity*, 3:1–18, 1993.
9. R. P. Brent. The parallel evaluation of arithmetic expressions in logarithmic time. In *Complexity of sequential and parallel numerical algorithms (Proc. Sympos., Carnegie-Mellon Univ., Pittsburgh, Pa., 1973)*, pages 83–102. Academic Press, New York, 1973.
10. P. Bürgisser. *Completeness and Reduction in Algebraic Complexity Theory*. Algorithms and Computation in Mathematics. Springer-Verlag, 2000.

11. P. Bürgisser, M. Clausen, and M. Shokrollahi. *Algebraic Complexity Theory*. Springer-Verlag, 1997.
12. H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. *Journal of Computer and System Sciences*, 57:200–212, 1998.
13. S. Istrail and D. Zivkovic. Bounded width polynomial size Boolean formulas compute exactly those functions in AC^0. *Information Processing Letters*, 50:211–216, 1994.
14. N. Limaye, M. Mahajan, and B. V. R. Rao. Arithmetizing classes around NC^1 and L. Technical Report 087, Electronic Colloquium on Computational Complexity (ECCC), 2007. Preliminary version Appeared in STACS 2007.
15. G. Malod. The complexity of polynomials and their coefficient functions. In *IEEE Conference on Computational Complexity*, pages 193–204, 2007.
16. G. Malod and N. Portier. Characterizing valiant's algebraic complexity classes. In *MFCS*, pages 704–716, 2006.
17. R. Raz. Multi-linear formulas for permanent and determinant are of super-polynomial size. In *STOC*, pages 633–641, 2004.
18. R. Raz. Multilinear-$NC^1 \neq$ multilinear-NC^2. In *FOCS*, pages 344–351, 2004.
19. R. Raz and A. Yehudayoff. Balancing syntactically multilinear arithmetic circuits. *Computational Complexity*, to appear.
20. P. M. Spira. On time hardware complexity tradeoffs for boolean functions. In *Fourth Hawaii International Symposium on System Sciences*, pages 525–527, 1971.
21. S.Toda. Counting problems computationally equivalent to the determinant. Technical Report CSIM 91-07, Dept. Comp. Sci. and Inf. Math., Univ. of Electro-Communications, Tokyo, 1991.
22. L. G. Valiant. Completeness classes in algebra. In *Symposium on Theory of Computing STOC*, pages 249–261, 1979.
23. L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. Fast parallel computation of polynomials using few processors. *SIAM J. Comput.*, 12(4):641–644, 1983.
24. H. Venkateswaran. Circuit definitions of nondeterministic complexity classes. *SIAM Journal on Computing*, 21:655–670, 1992.
25. H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer-Verlag New York Inc., 1999.