The arithmetic complexity of tensor contractions

Florent Capelli[1,2], Arnaud Durand[2], and Stefan Mengel*[3]

1 ENS Lyon, France
florent.capelli@ens-lyon.fr
2 IMJ UMR 7586 - Logique
Université Paris Diderot F-75205 Paris, France
durand@logique.jussieu.fr
3 Institute of Mathematics
University of Paderborn D-33098 Paderborn, Germany
smengel@mail.uni-paderborn.de

Abstract

We investigate the algebraic complexity of tensor calulus. We consider a generalization of iterated matrix product to tensors and show that the resulting formulas exactly capture VP, the class of polynomial families efficiently computable by arithmetic circuits. This gives a natural and robust characterization of this complexity class that despite its naturalness is not very well understood so far.

1998 ACM Subject Classification F.1.3 [Computation by abstract devices]: Complexity Measures and Classes — Relations among complexity measures

Keywords and phrases algebraic complexity, arithmetic circuits, tensor calculus

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.365

1 Introduction

The question of which polynomials can be computed by polynomial size arithmetic circuits is one of the central questions of algebraic complexity. It was first brought up explicitly by Valiant [11] who formulated a complexity theory in this setting with its own complexity classes and notions of completeness. Efficient computation in Valiant's model is formalized by the complexity class VP which consists of families of polynomials that can be computed by arithmetic circuits of polynomial size. Despite recent efforts relating VP to logically defined classes of polynomial families [9, 4], this class is not very well understood. This is reflected in the low number of helpful alternative characterizations and the conspicuous absence of any known natural complete problem.

Consequently, most progress in arithmetic circuit complexity has not been achieved by considering arithmetic circuits directly, but instead by considering the somewhat more restricted model of arithmetic branching programs (see e.g. [7, 11, 10, 5]). Arithmetic branching programs are widely conjectured to have expressivity strictly between that of arithmetic formulas and circuits, but have so far been better to handle with known proof techniques. One of the nice properties of branching programs that has often played a crucial role is that they can equivalently be seen as computing a specified entry of the iterated product of a polynomial number of matrices.

* Partially supported by DFG grants BU 1371/2-2 and BU 1371/3-1 and the [European Community's] Seventh Framework Programme [FP7/2007-2013] under grant agreement n° 238381.

We extend this view on branching programs by going from matrices to higher dimensional tensors. Consequently, we also go from matrix product to the generalized notion of contraction of tensors. It turns out that generalizing iterated matrix product to iterated tensor contractions does increase the expressive power of the model and that the resulting tensor formulas capture exactly VP. This characterization of VP turns out to be fairly robust in the sense that one can add different restrictions on the dimensions of the tensors without changing the expressive power of the model at all.

This is not the first time that the complexity of tensor calculus is studied. Damm, Holzer and McKenzie [3] and later Beaudry and Holzer [1] have characterized different boolean complexity classes by formulas having matrices as inputs and using addition, matrix product and tensor product as operations. Malod [6] adapted these formulas to the arithmetic circuit setting and showed characterizations for most arithmetic circuit classes. One difference between these results and those in our paper is that in [3, 6, 1] tensors are always encoded as matrices, i.e. the tensor product is expressed as the Kronecker product of two matrices. Another difference is that both the characterization of VP obtained in [6] and the similar characterization of LOGCFL (the Boolean analogon of VP) from [3] require an additional restriction, called *tameness* on the size of matrices computed at each gate of the formula. This restriction permits to control the growth of the intermediate objects in the computation but may seem not very natural. In this present work, working directly with tensors instead of a matrix representation makes such a restriction unnecessary and a more direct connection between VP and tensor calculus is established.

2 Preliminaries

Below, $\mathbb{K}$ is a field and bold letters denote tuples when there is no ambiguity on their length.

2.1 Arithmetic circuits

We will use the well known model of arithmetic circuits to measure the complexity of polynomials. In this section we give some definitions and well known properties of arithmetic circuits (see e.g. [2, 7] for more on the subject).

An *arithmetic circuit* is a directed acyclic graph with vertices of indegree 0 or 2 called *gates*. The gates of indegree 0 are called the *inputs* and are labeled with elements of $\mathbb{K}$ or variables. The gates of indegree 2, called *computation gates*, are labeled with operations of the field ($+$ and $\times$). The polynomial computed by a gate is defined inductively. The polynomial computed by an input gate is the one corresponding to its label. The polynomial computed by a computation gate is the sum or the product of the polynomials computed by its children. We assume that there exists a distinguished gate called the *output*. The polynomial computed by an arithmetic circuit is the one computed by its output gate. The *size* of a circuit C, denoted by $|C|$, is the number of vertices of its underlying DAG.

An arithmetic circuit C is said to be *skew* if for each $\times$-gate at least one of its children is an input of the circuit. A circuit is said to be *multiplicatively disjoint* if for each $\times$-gate, its two input subcircuits are disjoint.

A family $(f_n)_{n\in\mathbb{N}}$ of polynomials is in VP if there exists a family of multiplicatively disjoint circuits $(C_n)_{n\in\mathbb{N}}$ and a polynomial P such that for all $n \in \mathbb{N}$, C_n computes f_n and $|C_n| \leq P(n)$. The family (f_n) is in $\mathsf{VP}_{\mathrm{ws}}$ if the C_n are skew.

▶ Remark. Originally, VP was defined as families of polynomials that can be computed by polynomial size circuits and have polynomially bounded degree. As shown in [7] the definition given here is equivalent to the original one. We prefer this one here because the semantic condition on the degree is harder to deal with than multiplicatively disjointness which is more syntactic.

In the following, we will simulate arithmetic circuits by formulas computing tensors. We use the notion of *parse trees* of a circuit (see [7] for more details and bibliographical references on this notion). For a multiplicatively disjoint circuit C, we define its parse trees inductively. A parse tree T of C is a subgraph of C constructed as follows:

- Add the output of C to T
- For every gate v added to T do the following:
  - If v is a $+$-gate, add exactly one of its children to T.
  - If v is a $\times$-gate, add both of its children to T.

As C is multiplicatively disjoint, a parse tree of C is a tree. The monomial $m(T)$ computed by a parse tree T is the product of the labels of its leaves. The polynomial computed by C is the sum of the monomials of all parse trees of C.

2.2 Tensors

In this paper, we interpret tensors as multidimensional arrays. Their algebraic nature is not studied here. A good introduction to multilinear algebra and tensors can be found in [8].

Let $n_1, \ldots, n_k$ be k positive integers. A *k-dimensional tensor* T of *order* $(n_1, \ldots, n_k)$ is a mapping $T : [n_1] \times \ldots \times [n_k] \to \mathbb{K}$. For $i_1 \in [n_1], \ldots, i_k \in [n_k]$, we denote by $T[i_1, \ldots, i_k]$ the value of the mapping on the point $(i_1, \ldots, i_k)$. We call these values *entries* of T. We denote by $D(T)$ the *domain* of T. Obviously $D(T) = [n_1] \times \ldots \times [n_k]$. We also denote by $\dim(T)$ (equal to k, here), the dimension of T. The *size* of a tensor T, denote by $\|T\|$, is the number of entries, i.e. $\|T\| = \prod_{i=1}^{k} n_i$, where T is of order $(n_1, \ldots, n_k)$. The *maximal order* of T, denote by $\mathsf{maxorder}(T)$ is $\max_{i\in[k]} n_i$. In the following, we also call matrices (resp. vectors), tensors of dimension 2 (resp. 1).

▶ **Definition 1** (Contraction). Let T be a k-dimensional tensor of order $(n_1, \ldots, n_k)$ and G an l-dimensional tensor of order $(m_1, \ldots, m_l)$ with $k, l \geq 1$. If $n_k = m_1$, we denote by $T * G$ the *contraction* of T and G on the dimensions k and 1 which is a tensor of order $(n_1, \ldots, n_{k-1}, m_2, \ldots, m_l)$ defined as $(T * G)[\mathbf{e}_1, \mathbf{e}_2] = \sum_{i=1}^{n_k} T[\mathbf{e}_1, i]G[i, \mathbf{e}_2]$ for all $\mathbf{e}_1 \in [n_1] \times \ldots \times [n_{k-1}]$ and $\mathbf{e}_2 \in [m_2] \times \ldots \times [m_l]$.

▶ Remark. Obviously, contraction is a direct generalization of the matrix product. Indeed, if both T and G are matrices, then $T * G$ is the ordinary matrix product.

▶ **Proposition 2.** *Let T, G, H be tensors with $\dim(G) \geq 2$ such that $T * (G * H)$ and $(T * G) * H$ are both well defined. Then, $T * (G * H) = (T * G) * H$.*

Proof. By direct consequence of the associativity of the ordinary matrix product. For a tensor T of dimension $k \geq 2$ and order $(n_1, \ldots, n_k)$, and a tuple $\mathbf{e}$ of length $k-2$ we define the $n_1 \times n_k$ matrix $T_{\mathbf{e}} := (T[i, \mathbf{e}, j])_{i\in[n_1], j\in[n_k]}$. Then by associativity of matrix product:

$$\forall \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3 : T_{\mathbf{e}_1} * (G_{\mathbf{e}_2} * H_{\mathbf{e}_3}) = (T_{\mathbf{e}_1} * G_{\mathbf{e}_2}) * H_{\mathbf{e}_3}.$$

Hence, the claim follows when $\dim(T) \geq 2$ and $\dim(H) \geq 2$. For $\dim(T) = 1$ or $\dim(H) = 1$ the argument is similar. ◀

▶ Observation 3. If G is a vector, then the equality of Proposition 2 may not be true anymore. For example

$$\left(\begin{pmatrix}0 & 1\\0 & 0\end{pmatrix} * \begin{pmatrix}0\\1\end{pmatrix}\right) * \begin{pmatrix}1 & 0\\0 & 0\end{pmatrix} = \begin{pmatrix}1\\0\end{pmatrix} \neq \begin{pmatrix}0\\0\end{pmatrix} = \begin{pmatrix}0 & 1\\0 & 0\end{pmatrix} * \left(\begin{pmatrix}0\\1\end{pmatrix} * \begin{pmatrix}1 & 0\\0 & 0\end{pmatrix}\right).$$

where the input $\begin{pmatrix}0\\1\end{pmatrix}$ is a 1-dimensional tensor T of order (2) such that $T[1] = 0$ and $T[2] = 1$.

▶ **Definition 4.** A $\{*\}$-*formula* F is a labeled, ordered, rooted binary tree whose the leaves, called the inputs, are labeled by tensors whose entries are elements of $\mathbb{K}$ or variables and the other nodes are labeled by $*$. The tensor T_v computed by a node v is defined inductively:

- If v is a leaf then $T_v := \mathsf{label}(v)$.
- If v is labeled by $*$ and has left child v_1 and right child v_2 then $T_v := F_{v_1} * F_{v_2}$.

A $\{*\}$-formula computes the tensor computed by its root.

▶ Remark. In general, there may be a dimension mismatch in some contractions in an $\{*\}$-formula and thus the tensor computed by it may not be well-defined. However, detecting such $\{*\}$-formulas is easy and thus we only consider formulas in which no such problems occur.

As the entries of the input tensors are constants of $\mathbb{K}$ or variables, each entry of a tensor computed by a gate is a polynomial of $\mathbb{K}[X_1, \dots, X_n]$. This is why it makes sense to compare the computational power of $\{*\}$-formulas and arithmetic circuits defined in the last section. Moreover, of all the polynomials computed in the output of a $\{*\}$-formula we will mostly only be interested in one single polynomial. Thus we assume that the output tensor has only one single entry, i.e. the tensor is indeed a scalar. Observe that this form can always be achieved by contracting with vectors. We say that the scalar polynomial computed by a $\{*\}$-formula is *the* polynomial computed by it.

▶ **Definition 5.** The *size* of a $\{*\}$-formula F, denoted by $|F|$, is the number of $*$-gates plus the size of the inputs, i.e. $|F| := |\{v \mid \mathsf{label}(v) = *\}| + \sum_{T:T \text{ input of } F} \|T\|$. The *dimension* of F, denoted by $\dim(F)$ is the dimension of the tensor computed by F. The *maximal dimension* of F, denoted by $\operatorname{maxdim}(F)$ is the maximal dimension of the tensors computed at the gates of F, i.e. $\operatorname{maxdim}(F) := \max_{v\text{: gate in } F}(\dim(T_v))$. The *input dimension* of F is $\max_{v:v \text{ input of } F} \dim(T_v)$.

We will often mix the notations for tensors and for tensor formulas. For example, if F is a tensor formula computing the tensor T, we will speak of the order of F instead of T and write $F[\mathbf{e}]$ instead of $T[\mathbf{e}]$. Moreover, given two different formulas F and F', we will write $F \simeq F'$ if they compute the same tensor.

3 From arithmetic circuits to $\{*\}$-formulas

We describe how a family of polynomials in VP can be simulated by a family of $\{*\}$-formulas of polynomial size and maximal dimension 3. Our proof is inspired by a proof from [9] where it is shown that polynomials in VP can be represented by bounded treewidth CSPs.

▶ **Theorem 6.** *Let $(f_n) \in \mathsf{VP}$. There exists a family of $\{*\}$-formulas (F_n) of maximal dimension 3 and polynomial size such that F_n computes f_n for all n.*

We use the following observation from [9] which can be proved by combining results from Malod and Portier [7] and Valiant et al. [12].

▶ **Proposition 7.** *Let f be computed by an arithmetic circuit C of size s. Then there is an arithmetic circuit C' of size $s^{O(1)}$ that also computes f such that all parse trees of C' are isomorphic to a common tree $\mathcal{T}$.*

Theorem 6 follows direcly from Proposition 7 and the following lemma:

▶ **Lemma 8.** *Let C be an arithmetic circuit computing the polynomial f whose parse trees are all isomorphic to a common parse tree $\mathcal{T}$. Then there exists a $\{*\}$-formula F of maximal dimension 3 and of size $9|C|^3|\mathcal{T}|$ that computes f.*

Proof. We construct a tensor formula along the tree $\mathcal{T}$ which contains the sum of all monomials of f_n in its entries. We denote by $V(\mathcal{T})$ (resp. $V(C)$) the vertices of $\mathcal{T}$ (resp. C). For $s \in V(\mathcal{T})$, we call $\mathcal{T}_s$ the subtree of $\mathcal{T}$ rooted in s. We define a *partial parse tree* rooted in s to be a function $p : V(\mathcal{T}_s) \to V(C)$ respecting the following conditions for all $t \in V(\mathcal{T}_s)$:

1. If t is a leaf, then $p(t)$ is an input of C.
2. If t has one child t_1, $p(t)$ is a $+$-gate and $p(t_1)$ is a child of $p(t)$ in C.
3. If t has two children t_1 and t_2, then
   a. $p(t)$ is a $\times$-gate,
   b. $p(t_1)$ is the left child of $p(t)$, and
   c. $p(t_2)$ is the right child of $p(t)$.

We call these conditions the parse tree conditions. It is easy to see that when s is the root of $\mathcal{T}$ (and thus $\mathcal{T}_s = \mathcal{T}$) and $p : V(\mathcal{T}) \to V(C)$, then $p(V(\mathcal{T}))$ is the vertex set of a parse tree of C if and only if p is a partial parse tree rooted in s.

If p is a partial parse tree rooted in $s \in V(\mathcal{T})$, we define the monomial $m(p)$ computed by p by $m(p) := \prod_{t \in \mathsf{leaf}(\mathcal{T}_s)} \mathsf{label}(p(t))$. Observe that this is well defined as p respects, in particular, the first parse tree condition and thus $p(t)$ for $t \in \mathsf{leaf}(\mathcal{T}_s)$ is always an input of C. If p does not respect the parse tree conditions, we set $m(p) = 0$. With this notation we have

$$f = \sum_{p:V(\mathcal{T}) \to V(C)} m(p).$$

We index the vertices of C : $V(C) = \{v_1, \dots, v_r\}$ with $r = |C|$. We denote by E the tensor of dimension 1 and order (r) such that for all $i \le r$, $E[i] = 1$ and by $\delta_{i,j}$ the Kronecker function which equals 1 if $i = j$ and 0 otherwise. We construct by induction along the structure of $\mathcal{T}$ a $\{*\}$-formula F_s for each $s \in \mathcal{T}$. The formula F_s has dimension 2, order (r, r), size at most $9r^3|\mathcal{T}_s|$ and maximal dimension 3. Furthermore, for all $i, j \le r$:

$$F_s[i,j] = \delta_{i,j} \sum_{\substack{p:V(\mathcal{T}_s) \to V(C) \\ p(s)=v_i}} m(p).$$

Observing $f = E * F_s * E$ when s is the root of $\mathcal{T}$ completes the proof. We now describe the inductive construction of F_s. Several cases occur:

s is a leaf: In this case $\mathcal{T}_s$ consists only of the leaf s. The partial parse trees of $\mathcal{T}_s$ are functions $p : \{s\} \to V(C)$ and $m(p) = \mathsf{label}(p(s))$ if $p(s)$ is a input of C and $m(p) = 0$ otherwise. Then F_s consists of a $r \times r$ input matrix I such that for all $i, j \le r$,

$$I[i,j] = \begin{cases} \delta_{i,j}\mathsf{label}(v_j) \text{ if } v_j \text{ is an input} \\ 0 \text{ otherwise.} \end{cases}$$

Obviously, F_s is of size $r \le 9r^3$, of maximal dimension 2 and $I[i,j] = \delta_{i,j} \sum_{\substack{p:\{s\}\to V(C) \\ p(s)=v_i}} m(p)$.

s has one child s_1: We start with an observation on functions $p : V(\mathcal{T}_s) \to V(C)$. Let p_1 be the restriction of p on $V(\mathcal{T}_{s_1})$. If p is a partial parse tree, then p_1 is one, too, because it fulfills the parse tree conditions for all $t \in V(\mathcal{T}_{s_1}) \subseteq V(\mathcal{T}_s)$. Moreover, $m(p) = m(p_1)$ because the leaves in p and in p_1 are the same. In addition, if p is not a partial parse tree then

- either p violates a parse tree condition for $t \in \mathcal{T}_{s_1}$. In that case, p_1 is not a partial parse tree and then $m(p) = m(p_1) = 0$,
- or $p(s)$ is not a $+$-gate,
- or $p(s)$ is a $+$-gate but $p(s_1)$ is not a child of $p(s)$.

We encode these conditions in a tensor of dimension 3 and order (r, r, r) defined as

$$M[i,j,k] := \begin{cases} \delta_{j,k} \text{ if } v_j \text{ is } +\text{-gate and } v_i \text{ is a child of } v_j \\ 0 \text{ otherwise.} \end{cases}$$

Let $F_s := E * (F_{s_1} * M)$. Formula F_s is of maximal dimension 3, dimension 2 and order (r, r). We have $|F_s| \leq 9r^3(|\mathcal{T}_s| - 1) + r^3 + 2 + \|E\| \leq 9r^3|\mathcal{T}_s|$ and

$$\begin{aligned} F_s[j,k] &= \sum_{i=1}^{r}(\sum_{p=1}^{r} F_{s_1}[i,p]M[p,j,k]) \\ &= \delta_{j,k} \sum_{i=1}^{r}(\sum_{\substack{p_1 : V(\mathcal{T}_{s_1}) \to V(C) \\ p_1(s_1) = v_i}} m(p_1)M[i,j,j]). \end{aligned}$$

Let p be a function $p : V(\mathcal{T}_s) \to V(C)$ such that $p(s_1) = v_i$ and $p(s) = v_j$. Let p_1 be its restriction on $V(\mathcal{T}_{s_1})$. We have $m(p) = m(p_1)M[i,j,j]$, because

- if p is a partial parse tree then $M[i,j,j] = 1$ and thus $m(p_1)M[i,j,j] = m(p_1) = m(p)$,
- if v_j is not a $+$-gate then $m(p) = 0$ and also $m(p_1)M[i,j,j] = 0$ because $M[i,j,j] = 0$,
- if $p_1(s_1) = v_i$ is not a child of $v_j = p(s)$ then $m(p) = 0$. Since $M[i,j,j] = 0$, we have $m(p) = 0 = m(p_1)M[i,j,j]$.

This part of the proof is completed by remarking that, in each case

$$F_s[j,k] = \delta_{j,k} \sum_{i=1}^{r} \sum_{\substack{p_1 : V(\mathcal{T}_{s_1}) \to V(C) \\ p_1(s_1) = v_i}} m(p_1)M[i,j,k] = \delta_{j,k} \sum_{\substack{p : V(\mathcal{T}_s) \to V(C) \\ p(s) = v_j}} m(p).$$

s has two children, s_1 (left child) and s_2 (right child): As above, we encode the parse tree conditions in tensors of dimension 3 and contract them correctly to compute the desired result. This time there are two different tensors: one encoding the condition 3.b and one for 3.c. Let M_L and M_R be the two following (r, r, r) tensors:

$$M_L[i,j,k] = \begin{cases} \delta_{j,k} \text{ if } v_j \text{ is a } \times\text{-gate and } v_i \text{ is the left child of } v_j \\ 0 \text{ otherwise,} \end{cases}$$

$$M_R[i,j,k] = \begin{cases} \delta_{j,k} \text{ if } v_j \text{ is a } \times\text{-gate and } v_i \text{ is the right child of } v_j \\ 0 \text{ otherwise.} \end{cases}$$

Let F_s be the formula of maximal dimension 3, dimension 2 and order (r, r) defined as

$$F_s = (E * (F_{s_1} * M_L)) * (E * (F_{s_2} * M_R)).$$

We have $|F_s| = |F_{s_1}| + \|M_L\| + \|M_R\| + |F_{s_2}| + 5 + 2\|E\| \leq 9r^3|\mathcal{T}_s|$. In addition:

$$\begin{aligned} F_s[i,j] &= \sum_{k=1}^{r}((\sum_{a=1}^{r} F_{s_1}[a,a]M_L[a,i,k]) \times (\sum_{b=1}^{r} F_{s_2}[b,b]M_R[b,k,j])) \\ &= \delta_{i,j} \sum_{a,b=1}^{r} F_{s_1}[a,a]M_L[a,i,i]F_{s_2}[b,b]M_R[b,i,i] \\ &= \delta_{i,j} \sum_{a,b=1}^{r} \sum_{\substack{p_1 : V(\mathcal{T}_{s_1}) \to V(C) \\ p_1(s_1) = v_a}} \sum_{\substack{p_2 : V(\mathcal{T}_{s_2}) \to V(C) \\ p_2(s_2) = v_b}} m(p_1)m(p_2)M_L[a,i,i]M_R[b,i,i] \end{aligned}$$

Similarly as before, let $p : V(\mathcal{T}_s) \to V(C)$ such that $p(s) = v_i$, $p(s_1) = v_a$ and $p(s_2) = v_b$. We denote by p_1 (resp. p_2) the restriction of p on $V(\mathcal{T}_{s_1})$ (resp. $V(\mathcal{T}_{s_2})$) and show that

$$m(p) = M_L[a, i, i]M_R[b, i, i]m(p_1)m(p_2)$$

by studying the possible cases:

- If p is a partial parse tree then p_1 and p_2 are, too. Moreover, since s has two children, $p(s) = v_i$ is necessarily a $\times$-gate, v_a its left child and v_b its right child. It follows that $M_L[a, i, i] = M_R[b, i, i] = 1$ and

$$m(p) = \prod_{l \in \mathsf{leaf}(\mathcal{T}_s)} \mathsf{label}(l) = \prod_{l \in \mathsf{leaf}(\mathcal{T}_{s_1})} \mathsf{label}(l) \prod_{l \in \mathsf{leaf}(\mathcal{T}_{s_2})} \mathsf{label}(l) = m(p_1)m(p_2).$$

- If p is not a partial parse tree then three cases can occur: If p_1 (resp. p_2) is not a partial parse tree, then $m(p_1) = 0$ (resp. $m(p_2) = 0$). If v_i is not a $\times$-gate, then $M_L[a, i, i] = 0$. Finally, if v_a (resp. v_b) is not the left (resp. right) child of v_i, then $M_L[a, i, i] = 0$ (resp. $M_R[b, i, i] = 0$). In all those cases, $M_L[a, i, i]M_R[b, i, i]m(p_1)m(p_2) = 0 = m(p)$.

This completes the proof. ◀

4 From $\{*\}$-formulas to arithmetic circuits

In this section we will show that the polynomials computed by polynomial size $\{*\}$-formulas can also be computed by polynomial size arithmetic circuits. We start by first proving this for formulas with bounded maximal dimension. Then we extend this result by showing that any $\{*\}$-formula can be transformed into an equivalent one with bounded maximal dimension without increasing the size.

4.1 Formulas with bounded maximal dimension

▶ **Proposition 9.** *Let F be a $\{*\}$-formula of maximal dimension k, dimension $l \leq k$ and order $(n_1, \dots, n_l)$. Let $n := \max_{T:T \text{ input of } F}(\mathsf{maxorder}(T))$. Then there exists a multiplicatively disjoint circuit C of size at most $2n^{k+1}|F|$ such that for all $\mathbf{e} \in D(F)$ there exists a gate $v_{\mathbf{e}}$ in C computing $F[\mathbf{e}]$.*

Proof. If F is an input, let C be the circuit having $\prod_{i=1}^{l} n_i$ inputs, each one labeled with an entry of F. The size of C is $\prod_{i=1}^{l} n_i \leq n^k$.

If $F = G * H$, by induction we have circuits C_G and C_H with the desired properties for G and H. The dimension of F is less than k and for $\mathbf{e} \in D(F)$, $F[\mathbf{e}] = \sum_{i=1}^{m} G[\mathbf{e}_1, i]H[i, \mathbf{e}_2]$ with $m \leq n$.

Each $G[\mathbf{e}_1, i]$ and $H[i, \mathbf{e}_2]$ is computed by a gate of C_G and C_H,respectively, so we can compute $F[\mathbf{e}]$ by adding at most $2n$ gates (m $\times$-gates and $m - 1$ $+$-gates). As there are at most n^k entries in F, we can compute all of them with a circuit C by adding at most $2n \times n^k$ gates to $C_H \cup C_G$.

The circuit C is multiplicatively disjoint since each $\times$-gate receives one of its input from C_G and the other one from C_H. Also $|C| = |C_G| + |C_H| + 2n^{k+1} \leq 2n^{k+1}|F|$. ◀

▶ **Corollary 10.** *Let (F_n) be a family of $\{*\}$-formulas of polynomial size and of maximal dimension k computing a family (f_n) of polynomials. Then (f_n) is in* VP.

4.2 Unbounded maximal dimension

Since the size of the circuit constructed in the previous section is exponential in $k := \mathrm{maxdim}(F)$, we cannot apply the results from there directly if k is not bounded by a constant. Somewhat surprisingly we will see in this section that one does not gain any expressivity by letting intermediate dimensions of formulas grow arbitrarily. Thus bounding $\mathrm{maxdim}(F)$ is not a restriction of the computational power of $\{*\}$-formulas.

▶ **Definition 11.** A $\{*\}$-formula F of dimension k and input dimension p is said to be *tame* if $\mathrm{maxdim}(F) \leq \max(k, p)$.

▶ **Definition 12.** A $\{*\}$-formula F is said to be *totally tame* if each subformula of F is tame.

Let us remark again that also in [3] and [6] there is a notion of tameness that prevents intermediate results from growing too much during the computation. It turns out that in those papers tameness plays a crucial role: Tame formulas can be evaluated efficiently while general formulas are hard to evaluate in the respective models. We will see that in our setting tameness is not a restriction at all. Indeed, any $\{*\}$-formula can be turned into an equivalent totally tame formula without any increase of its size. Thus totally tame and general formulas have the same expressive power in our setting which is a striking difference to the setting from [3] and [6]. We start with the following lemma:

▶ **Lemma 13.** *Let F be a totally tame formula with $\dim(F) = k$ and input dimension p. For all totally tame formulas E of dimension 1 and input dimension at most p, there exist totally tame formulas G_r and G_l of size $|F * E| = |E * F| = |F| + |E| + 1$ such that $G_r \simeq F * E$ and $G_l \simeq E * F$.*

Proof. We only show the construction of G_r; the construction of G_l is completely analogous. We proceed by induction on F.

If F is an input, then $\mathrm{maxdim}(F) = \dim(F) = p$. Let E be any totally tame formula of dimension 1 and input dimension at most p. We set $G_r := F*E$. Clearly, $k = \dim(G_r) = p-1$. Furthermore, $\mathrm{maxdim}(G_r) = \max(p-1, \mathrm{maxdim}(F), \mathrm{maxdim}(E)) \leq p$ because E has input dimension at most p and is totally tame. Thus G_r is totally tame.

Let now $F = F_1 * F_2$. Let $k_1 := \dim(F_1)$ and $k_2 := \dim(F_2)$. Let E be a totally tame formula of dimension 1 and input dimension at most p.

- If $\dim(F_2) = 1$, we claim that $G_r = F * E$ is totally tame. Indeed, since $\dim(F_2) = 1$, we have $\dim(F) = k_1 - 1$. But F is by assumption tame, so $k_1 = \dim(F_1) \leq \max(k_1 - 1, p)$. Hence $k_1 \leq p$ and $\dim(F) \leq p$. Thus all intermediate results of G_r have dimension at most p, so G_r is tame. But then it is also totally tame, because its subformulas are totally tame by assumption.
- If $\dim(F_2) = 2$, we have $p \geq 2$ obviously and $\dim(F) = \dim(F_1)$. F_2 is a subformula of F, so it is totally tame, too. Furthermore, $F_2 * E$ is of dimension 1 and it is also totally tame since $2 \leq p$. Moreover, by Proposition 2 we have $F * E \simeq F_1 * (F_2 * E)$. Applying the induction hypothesis on F_1 and $(F_2 * E)$ gives the desired G_r.
- If $\dim(F_2) > 2$, by Proposition 2 we have $F * E \simeq F_1 * (F_2 * E)$. We first apply the induction hypothesis on F_2 and E to construct a totally tame formula G' computing $F_2 * E$. Finally $G_r := F_1 * G'$ is totally tame since F_1 and G' are totally tame and F is of dimension $k = k_1 + k_2 - 2 \geq \max(k_1, k_2 - 1)$.

◀

We now prove the main proposition of this section.

▶ **Proposition 14.** *For every $\{*\}$-formula F there exists a totally tame $\{*\}$-formula F' such that $F' \simeq F$ and $|F| = |F'|$.*

Proof. The proof is by induction on F. If F is an input then it is trivially totally tame as the dimension of F is equal to the input dimension of F. So we set $F' := F$.

If $F = F_1 * F_2$ then several cases can occur depending on the dimension of F_1 and F_2. We denote by k, k_1, k_2 the dimensions of F, F_1 and F_2 respectively. We recall that $k = k_1 + k_2 - 2$.

- If both k_1 and k_2 are different from 1. Then $F' = F'_1 * F'_2$ is totally tame since $k \geq \max(k_1, k_2)$
- If $k_1 = 1$ or $k_2 = 1$, we use Lemma 13 on F'_1 and F'_2 to construct F' of size $|F|$, totally tame, computing $F_1 * F_2$.

◀

Combining Proposition 14 and Corollary 10 we get the following theorem:

▶ **Theorem 15.** *Let (F_n) be a family of $\{*\}$-formulas of polynomial size and input dimension p (independent of n) computing a family of polynomials (f_n). Then (f_n) is in* VP.

Proof. Applying Proposition 14 on (F_n) gives a family (F'_n) computing (f_n) such that (F'_n) is tame. Then the maximal dimension of F'_n is p (because F_n is scalar, thus of dimension 1) and applying Corollary 10 proves the claim. ◀

4.3 Unbounded input dimension

While we got rid of the restriction on the maximum dimension of $\{*\}$-formulas in the last section, we still have a bound on the dimension of the inputs in Theorem 15. In this section we will show that this bound is not necessary to have containment of the computed polynomials in VP. We will show that inputs having "big" dimension can be computed by polynomial size $\{*\}$-formulas of input dimension 3. We can then use this to eliminate inputs of dimension more than 3 in $\{*\}$-formulas. Applying Theorem 15 we conclude that the only restriction on $\{*\}$-formulas that we need to ensure containment in VP is the polynomial size bound.

▶ **Proposition 16.** *Let T be a r-dimensional tensor of order $(n_1, \ldots, n_r)$. Let $L := \|T\| = \prod_{i=1}^{r} n_i$ be the number of entries in T. Then there is a $\{*\}$-formula F of size $r + 1 + L^3 + 2L$ and input dimension 3 computing T.*

Proof (sketch). Choose an arbitrary bijection $B : [L] \to [n_1] \times \ldots \times [n_r]$. Let furthermore $B_i : [L] \to [n_i]$ for $i \leq r$ be the projection of B onto the i-th coordinate. We define the 3-dimensional tensors T_i of order (L, n_i, L) by

$$T_1[m,k,n] = \begin{cases} T[B(m)] \text{ if } m = n \text{ and } B_1(m) = k \\ 0 \text{ otherwise.} \end{cases}$$

and, for $2 \leq i \leq r$,

$$T_i[m,k,n] = \begin{cases} 1 \text{ if } m = n \text{ and } B_i(m) = k \\ 0 \text{ otherwise.} \end{cases}$$

By induction one can show that for the tensor $P = T_1 * \ldots * T_r$, $P[m, k_1, \ldots, k_r, n] = T[k_1, \ldots, k_r]$ holds if $m = n$ and $B(m) = (k_1, \ldots, k_r)$ and $P[m, k_1, \ldots, k_r, n] = 0$ hold otherwise. Hence $T = E * P * E$ where E is a vector of order L filled with 1. ◀

The following theorem is a direct consequence of Proposition 16 and Theorem 15.

▶ **Theorem 17.** *Let (F_n) be a family of $\{*\}$-formulas of polynomial size computing a family of polynomials (f_n). Then f_n is in* VP.

5 The power of contracting with vectors

In this section we will make a finer examination of where exactly the additional power originates when going from iterated matrix product of [7] to tensor contractions. We will see that this additional expressivity crucially depends on the possiblity of contracting tensors on more than two of their dimensions. We will show that when we prevent this possibility by disallowing contractions with vectors – which are used in the proof of Theorem 6 to "collapse" dimensions not needed anymore so that we can access other dimensions to contract on – the expressivity of $\{*\}$-formulas drops to that of iterated matrix product.

Observe that we cannot assume that $\{*\}$-formulas compute scalars in this setting, because we cannot decrease the dimension of the tensors computed by a formula. Also we cannot compute all entries of the output at the same time efficiently, because there might be exponentially many such entries. But we will see in the following Proposition that we can compute each individual entry of the output more efficiently than in the general setting where contraction with tensors is allowed.

▶ **Proposition 18.** *Let F be a $\{*\}$-formula of order $(n_1, \dots, n_k)$ whose inputs are all of dimension at least 2. Then for all $\mathbf{e} \in D(F)$ there exists a skew arithmetic circuit C of size at most $2n^3|F|$ where $n := \max_{T\colon T \text{ input of } F}(\mathsf{maxorder}(T))$ computing $F[\mathbf{e}]$.*

Proof. By Proposition 2 we can write F as $A_1 * (A_2 * (A_3 * \dots * A_n))$. The proof then follows easily by induction: We do the same construction as in Theorem 10 but this time we only have n^2 entries and at each $*$-gate, one side is an input, resulting in a skew circuit. ◀

The case of Proposition 18 exactly corresponds to the characterization of $\mathsf{VP}_{\mathrm{ws}}$ by Malod and Portier [7] by n products of matrices of size $n \times n$. Thus Proposition 18 naturally generalizes this result and the real new power seen in Theorem 6 must come from the use of vectors in the products. As we have seen in the proof of Proposition 18 it is crucial that vectors are the only case which breaks the associativity of Proposition 2. So what looked like a not very important edge case in Observation 3 plays a surprisingly important role for the expressivity of $\{*\}$-formulas.

6 The $*_{i,j}$ operators

The $*$-operator contracts tensors only in a very specific way: It always only contracts on the last dimension of one tensor and the first dimension of the other one. It is thus natural to ask if this is a restriction of the computational power of the formulas. In this section we will see that it is indeed not. If we allow free choice of the dimensions to contract on during a contraction this does not make the resulting polynomials harder to compute. To formalize this we give the following definition of a contraction $*_{i,j}$.

▶ **Definition 19.** Let T be a k-dimensional tensor of order $(n_1, \dots, n_k)$ and G a l-dimensional tensor of order $(m_1, \dots, m_l)$ with $k, l \geq 1$. When $n_i = m_j$ for $i \leq k$ and $j \leq l$, we denote by $T *_{i,j} G$ the contraction of T and G on the dimensions i and j the $(k + l - 2)$-dimensional tensor of order $(n_1, \dots, n_{i-1}, n_{i+1}, \dots, n_k, m_1, \dots, m_{j-1}, m_{j+1}, \dots, m_l)$ defined as

$$(T *_{i,j} G)[\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_4] = \sum_{r=1}^{n_i} T[\mathbf{e}_1, r, \mathbf{e}_2] G[\mathbf{e}_3, r, \mathbf{e}_4]$$

for all $\mathbf{e}_1 \in [n_1] \times \dots \times [n_{i-1}]$, $\mathbf{e}_2 \in [n_{i+1}] \times \dots \times [n_k]$, $\mathbf{e}_3 \in [m_1] \times \dots \times [m_{j-1}]$ and $\mathbf{e}_4 \in [m_{j+1}] \times \dots \times [m_l]$.

$\{*_{i,j}\}$-*formulas* are defined in complete analogy to $\{*\}$-formulas.

It turns out that $\{*_{i,j}\}$-formulas cannot compute more than $\{*\}$-formulas, so the free choice of the dimensions to meld on does not change much.

▶ **Theorem 20.** *Let (F_n) be a family of $\{*_{i,j}\}$-formulas of polynomial size computing a family of polynomials (f_n). Then (f_n) is in* VP.

The proof of Theorem 20 follows a similar approach as that of Theorem 17. Let us sketch some key steps here: If we bound the maximal dimension of $\{*_{i,j}\}$-formulas by a constant k, it is easy to see that the proof of Theorem 10 can be adapted to $\{*_{i,j}\}$-formulas in a straightforward way. The main complication is then turning general $\{*_{i,j}\}$-formulas into totally tame ones. $*_{i,j}$ is not associative anymore, and this makes a straightforward translation of the proof of Proposition 14 tricky. These problems can be solved by the observation that the crucial steps in the process of making a formula tame are those where a $\{*_{i,j}\}$-formula is multiplied by a tensor of dimension 1. But for such contractions we can give explicit formulas for different cases that may occur, so again every $\{*_{i,j}\}$-formula has an equivalent tame $\{*_{i,j}\}$-formula. Combining this with Proposition 16 completes the proof.

7 Conclusion

We have shown that one can get a robust characterization of VP by formulas with tensors as input and tensor contraction as the only operation. This generalizes the known characterization of $\mathsf{VP}_{\mathrm{ws}}$ by iterated matrix product by Malod and Portier [7]. In some aspects the situation in our setting is more subtle, though. We remarked that vectors and in general breaking associativity plays a crucial role if we want to characterize VP. Also, unlike for iterated matrix product we have to make a choice if we take $*_{i,j}$ or $*$ as our basic operation. It is easy to check that using the equivalence to $*_{i,j}$ for matrix product would merely be transposing the matrix, so it clearly does not change the expressivity of the model. But fortunately also in our setting, the choice of $*_{i,j}$ or $*$ does not influence the complexity of the computed polynomials.

Unfortunately, unlike for iterated matrix product our characterization seemingly does not directly lead to a characterization of VP by something similar to branching programs. We still think that such a characterization is highly desirable, because the branching program characterization of $\mathsf{VP}_{\mathrm{ws}}$ has been the source of important insights in arithmetic circuit complexity. Thus we believe that a similar characterization of VP might lead to a better understanding of VP, a class that is arguably not very well understood, yet.

Let us quickly discuss several extensions to the results in this paper that we had to leave out for lack of space: First, analyzing the proofs of Section 4 a little more carefully one can see that our results remain true if one does not measure the size of a tensor as the number of its entries but as the number of its *nonzero* entries. This makes it possible to allow inputs of large dimension and large order.

Also, it seems plausible and straightforward to generalize our results to arbitrary semi-rings in the style of Damm, Holzer and McKenzie [3]. Choosing different semi-rings one would then probably get characterizations of classes like LOGCFL and its counting, mod-counting and gap-versions. The main new consideration would be the treatment of uniformity in these settings which appears to be possible with a more refined analysis of our proofs.

Finally, for tensors there are other natural operations to perform on them like addition or tensor product. It is natural to ask, if adding such operations does change the complexity of the resulting polynomials. While it is straightforward to see that adding only tensor product as an operation does not increase the expressivity of $\{*\}$-formulas, we could so far not answer the corresponding question for addition. Therefore, we leave this as an open question.

Acknowledgements

We thank Yann Strozecki for a detailed and helpful feedback on an early version of this paper. We also thank Hervé Fournier, Guillaume Malod and Sylvain Perifel for helpful discussions.

References

1 Martin Beaudry and Markus Holzer. The complexity of tensor circuit evaluation. *Computational Complexity*, 16(1):60–111, 2007.

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

3 C. Damm, M. Holzer, and P. McKenzie. The complexity of tensor calculus. *Computational Complexity*, 11(1-2):54–89, 2002.

4 A. Durand and S. Mengel. The Complexity of Weighted Counting for Acyclic Conjunctive Queries. *Arxiv preprint arXiv:1110.4201*, 2011.

5 P. Koiran. Arithmetic circuits: The chasm at depth four gets wider. *Theoretical Computer Science*, 448(0):56 – 65, 2012.

6 G. Malod. Circuits arithmétiques et calculs tensoriels. *Journal of the Institute of Mathematics of Jussieu*, 7:869–893, 2005.

7 G. Malod and N. Portier. Characterizing Valiant's algebraic complexity classes. *Journal of complexity*, 24(1):16–38, 2008.

8 M. Marcus. *Finite dimensional multilinear algebra*, volume 1. M. Dekker, 1973.

9 S. Mengel. Characterizing Arithmetic Circuit Classes by Constraint Satisfaction Problems - (Extended Abstract). In *ICALP (1)*, pages 700–711, 2011.

10 N. Nisan. Lower bounds for non-commutative computation. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 410–418. ACM, 1991.

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

12 L.G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. Fast parallel computation of polynomials using few processors. *SIAM Journal on Computing*, 12:641, 1983.