The Complexity of Tensor Calculus

Carsten Damm*
Institut für Angewandte und Numerische Mathematik,
Georg-August-Universität Göttingen,
D-37083 Göttingen, Germany
email: damm@math.uni-goettingen.de

Markus Holzer† and Pierre McKenzie†
Département d'I.R.O., Université de Montréal, C.P. 6128,
succ. Centre-Ville, Montréal (Québec), H3C 3J7 Canada
email: {holzer,mckenzie}@iro.umontreal.ca

Abstract

Tensor calculus over semirings is shown relevant to complexity theory in unexpected ways. First, evaluating well-formed tensor formulas with explicit tensor entries is shown complete for $\oplus$P, for NP, and for #P as the semiring varies. Indeed the permanent of a matrix is shown expressible as the value of a tensor formula in much the same way that Berkowitz' theorem expresses its determinant. Second, restricted tensor formulas are shown to capture the classes LOGCFL and NL, their parity counterparts $\oplus$LOGCFL and $\oplus$L, and several other counting classes. Finally, the known inclusions NP/*poly* $\subseteq$ $\oplus$P/*poly*, LOGCFL/*poly* $\subseteq$ $\oplus$LOGCFL/*poly*, and NL/*poly* $\subseteq$ $\oplus$L/*poly*, which have scattered proofs in the literature [21, 39], are shown to follow from the new characterizations in a single blow.

1 Introduction

Consider an algebraic structure $\mathcal{S}$ with certain operations. The following problem is sometimes called the *word problem of* $\mathcal{S}$: given a reasonable encoding of a well-formed expression T over $\mathcal{S}$, and an "accepting" subset of $\mathcal{S}$ (given explicitly by enumeration or implicitly by a condition), decide whether T evaluates to an element in the accepting set. Many special cases of this problem have been studied. For example, intricate NC^1 upper bounds are known when $\mathcal{S}$ is the Boolean [13] or a more general semiring [14], the case of groups was crucial

*Partially supported by Deutsche Forschungsgemeinschaft grant Me 1077/14-1. Part of the work was done while the author was at Universität Trier, Germany.

†Supported by the Québec FCAR and by the NSERC of Canada.

to elucidating the power of bounded-width branching programs [6], the case of groupoids was shown to capture LOGCFL [7], the case of Boolean and integer matrices relates to NL and to the determinant [3, 10, 16, 24, 36, 41], the case of $\mathcal{F}$-matrices for various fields $\mathcal{F}$ captures linear algebra and counting classes like $\oplus$L [11, 15], and the case of sets of natural numbers with union and sum as operations captures NP [34].

In this paper we study the word problem generalized to *multilinear algebra* expressed by tensors. This is interesting for two reasons. First, tensors have many applications in physics, engineering, and computer science [18, 35], and it is natural to investigate the complexity of their handling. Second, we show that basic tensor calculus not only captures natural complexity classes in simple ways, but it yields simpler unified proofs of non-uniform inclusions formerly scattered in the literature.

Our results concern tensors over semirings, including the Boolean semiring $\mathbb{B} = (\{0,1\}, \vee, \wedge)$, the fields of characteristic 2, and the natural numbers $\mathbb{N} = (\{0,1,\dots\}, +, \cdot)$. Tensors are "multi-dimensional matrices," i.e., cuboid-like arrangements of semiring elements. Their algebraic interpretation, which need not concern us here, gives rise to three natural operations: the sum $(+)$ of two tensors of identical dimensions, the product $(\otimes)$ of two arbitrary tensors, and the junction of a tensor with suitable dimensions (see Appendix A for details). Handling tensors numerically routinely requires the evaluation of tensor expressions involving ordinary matrix operations as well as $+$ and $\otimes$. A special case of such an evaluation process occurs when the resulting tensor is in fact a scalar and it is only required to determine whether this scalar equals the zero of $\mathcal{S}$. We call the latter problem the *non-zero tensor problem* $0 \neq \mathrm{val}_{\mathcal{S}}$, which is central to our study. Its complexity can be regarded as intrinsic for the tensor calculus over the specified semiring. Our precise characterizations are as follows:

- A polytime Turing machine exists which, given a matrix A over $\mathbb{N}$, produces a tensor formula that evaluates to the permanent of A; this resembles Berkowitz' theorem giving a similar formula for the determinant.
- Evaluating a tensor formula over the natural numbers is #P-complete under polytime Turing reductions.
- $0 \neq \mathrm{val}_{\mathbb{B}}$ and $0 \neq \mathrm{val}_{\mathbb{F}_2}$ are respectively NP-complete and $\oplus$P-complete under polytime many-one reductions.
- By natural syntactic restrictions on the admissible formula types, we obtain problems that are complete under many-one logspace reductions for the classes LOGCFL and NL, and their parity counterparts $\oplus$LOGCFL and $\oplus$L, respectively; similar statements hold for tensors over other rings and the corresponding counting classes.

To illustrate the strength of our characterizations, we are able to give a very intuitive and *unified* justification for non-uniform simulations of Boolean computations by parity computations. The existence of such simulations had been proved before in several papers. NP/*poly* $\subseteq \oplus$P/*poly*, is a corollary of a randomized reduction of NP to UP proved by Valiant and Vazirani [39]. Wigderson [43] proved the logspace analogue on the basis of the Isolating Lemma of [29] and shortly thereafter, by a similar technique, Gál [20] proved the corresponding result for LOGCFL. In [9] a generalization and more efficient simulation for NL/*poly* $\subseteq \oplus$L/*poly* was given. Our completeness results allow extending ideas from the latter paper to also cover the other simulations, basically using a single proof.

The structure of the paper is as follows. In the next section we introduce the terminology and basic parsing techniques for tensor formulas. We also introduce the complexity classes needed in later sections. In Section 3 we give a polytime algorithm to construct a tensor formula which evaluates to the permanent of a square matrix. The structure of this formula is very regular and depends only on the input size, not on the matrix itself. Then in Section 4 we introduce algebraic Turing machines, a natural generalization of arithmetic branching programs as introduced in [9], and use this model in Section 5 to prove our completeness results. In Section 6 we give a unified and intuitive proof for the existence of nonuniform simulations between Boolean and arithmetic complexity classes. Finally, in the last section, we conclude and discuss related results.

To support intuition we strictly base all our formalism on matrices rather than tensors. However, this is only a different terminology for the same thing. The connection between the multi-index tensor notation and the two-index matrix notation is well-known [18]. It is just a bijection between entries of tensors and corresponding matrices and is explained in Appendix A.

2 Definitions, Notations, and Basic Techniques

We use standard notation from computational complexity, such as contained in, e.g., [5, 42]. In particular we recall the inclusion chain: NL $\subseteq$ LOGCFL $\subseteq$ P $\subseteq$ NP. Here NL (NP, respectively) denotes the set of problems solvable by nondeterministic Turing machines in logspace (polytime, respectively), and LOGCFL is the class of all languages accepted by nondeterministic auxiliary pushdown automata working in polytime and logspace. Moreover, P denotes the class of problems solvable by deterministic Turing machines in polytime. The corresponding counting versions of NL, LOGCFL, and NP, denoted by #L, #LOGCFL, and #P, are the classes of functions f, such that there is a machine M with the same resources as the underlying base class, such that $f(x)$ equals the number of accepting computations of M on x (see [4, 38, 41]). Decision classes like NL, LOGCFL, and NP are defined on *Boolean* computation models, in that they rely on the mere *existence* of accepting computations. If existence is replaced by the predicate "there is an *odd number of* accepting computations," we obtain the parity versions $\oplus$L, $\oplus$LOGCFL, and $\oplus$P introduced in [20, 27, 31]. More formally, $\oplus$L is the class of sets of type $\{\, x \mid f(x) \neq 0 \pmod 2 \,\}$ for some $f \in$ #L. The classes $\oplus$LOGCFL and $\oplus$P are defined analogously. These classes were intensively studied in the literature, see, e.g., [11, 21, 37]. The classes MOD_q-L, MOD_q-LOGCFL, and MOD_q-P are defined similarly with respect to counting modulo q. Finally, consider the following complexity classes $\textit{Gap}\text{P} = \{ f - g \mid f, g \in \#\text{P} \}$, $\textit{Gap}\text{LOGCFL} = \{ f - g \mid f, g \in \#\text{LOGCFL} \}$, and $\textit{Gap}\text{L} = \{ f - g \mid f, g \in \#\text{L} \}$. For details on these classes we refer to [19].

A *semiring* is a tuple $(\mathcal{S}, +, \cdot)$ with $\{0, 1\} \subseteq \mathcal{S}$ and binary operations $+, \cdot : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ (sum and product), such that $(\mathcal{S}, +, 0)$ is a commutative monoid, $(\mathcal{S}, \cdot, 1)$ is a monoid, multiplication distributes over sum, and $0 \cdot a = a \cdot 0 = 0$ for every a in $\mathcal{S}$ (see, e.g., [26]). A semiring is *commutative* if and only if $a \cdot b = b \cdot a$ for every a and b, and it is *finitely generated* if there is a finite set $\mathcal{G} \subseteq \mathcal{S}$ generating all of $\mathcal{S}$ by summation. The special choice of $\mathcal{G}$ has no influence on the complexity of problems we study in this paper. Throughout the paper we consider the following semirings: the Booleans $(\mathbb{B}, \vee, \wedge)$, finite fields $\mathbb{F}_{2^d}$ of characteristic 2 (although all our considerations can be generalized to any finite field), residue class rings $(\mathbb{Z}_q, +, \cdot)$, the naturals $(\mathbb{N}, +, \cdot)$, or the integers $\mathbb{Z} = (\mathbb{Z}, +, \cdot)$.

Let $\mathbb{M}_{\mathcal{S}}$ denote the set of all *matrices* over $\mathcal{S}$, and define $\mathbb{M}_{\mathcal{S}}^{k,\ell} \subseteq \mathbb{M}_{\mathcal{S}}$ to be the set of all *matrices of order* $k \times \ell$. For a matrix A in $\mathbb{M}_{\mathcal{S}}^{k,\ell}$ let $I(A) = [k] \times [\ell]$, where $[k]$ denotes the set $\{1, 2, \ldots, k\}$. The (i, j)th entry of A is denoted by $a_{i,j}$ or $(A)_{i,j}$. Addition and multiplication of matrices in $\mathbb{M}_{\mathcal{S}}$ are defined in the usual way. Additionally we consider the *tensor product* $\otimes : \mathbb{M}_{\mathcal{S}} \times \mathbb{M}_{\mathcal{S}} \to \mathbb{M}_{\mathcal{S}}$ of matrices, also known as Kronecker product, outer product, or direct product, which is defined as follows: for $A \in \mathbb{M}_{\mathcal{S}}^{k,\ell}$ and $B \in \mathbb{M}_{\mathcal{S}}^{m,n}$ let $A \otimes B \in \mathbb{M}_{\mathcal{S}}^{km,\ell n}$ be

$$A \otimes B := \begin{pmatrix} a_{1,1} \cdot B & \ldots & a_{1,\ell} \cdot B \\ \vdots & \ddots & \vdots \\ a_{k,1} \cdot B & \ldots & a_{k,\ell} \cdot B \end{pmatrix}.$$

Hence $(A \otimes B)_{i,j} = (A)_{q,r} \cdot (B)_{s,t}$ where $i = k \cdot (q-1) + s$ and $j = \ell \cdot (r-1) + t$.

The following notation is heavily used: let I_n be the order n identity matrix, e_i^n the ith unit row vector of length n, D_i^n the $n \times n$ "dot matrix" having one in position (i, i) and zeros elsewhere, and let $A^{\otimes n}$ stand for the n-fold iteration $A \otimes A \otimes \cdots \otimes A$. Further, we will make use of the following identities:

Proposition 1. *The following hold when the expressions are defined [33]:*

1. $(A \otimes B) \otimes C = A \otimes (B \otimes C)$.
2. $(A + B) \otimes (C + D) = A \otimes C + A \otimes D + B \otimes C + B \otimes D$.
3. $(A \otimes B) \cdot (C \otimes D) = (A \cdot C) \otimes (B \cdot D)$.
4. $(A \cdot B)^{\otimes n} = A^{\otimes n} \cdot B^{\otimes n}$ *for any* $n \geq 1$.

Now we are ready to define tensor formulas over semirings.

Definition 2. *The* tensor formulas *over a semiring* $\mathcal{S}$ *and their* order *are recursively defined as follows:*

1. *Every matrix F from $\mathbb{M}_{\mathcal{S}}^{k,\ell}$ with entries from $\mathcal{S}$ is a (*atomic*) tensor formula of order $k \times \ell$.*

2. *If F and G are tensor formulas of order $k \times \ell$ and $m \times n$, respectively, then*

   (a) $(F + G)$ is a tensor formula of order is $k \times \ell$ if $k = m$ and $\ell = n$.

   (b) $(F \cdot G)$ is a tensor formula of order $k \times n$ if $\ell = m$.

   (c) $(F \otimes G)$ is a tensor formula of order $km \times \ell n$.

3. *Nothing else is a tensor formula.*

For a tensor formula F of order $k \times \ell$ let $I(F) = [k] \times [\ell]$ be its "set of indices." Let $\mathbb{T}_{\mathcal{S}}$ denote the set of all tensor formulas over $\mathcal{S}$, and define $\mathbb{T}_{\mathcal{S}}^{k,\ell} \subseteq \mathbb{T}_{\mathcal{S}}$ to be the set of all tensor formulas of order $k \times \ell$.

In this paper we only consider finitely generated $\mathcal{S}$, and we assume that atomic tensor formula entries are from $\mathcal{G} \cup \{0\}$, where $\mathcal{G}$ is a generating set of $\mathcal{S}$. Hence, atomic tensor formulas, i.e., matrices, can be string-encoded using list notation such as "[[001][101]]." Non-atomic tensor formulas can be encoded over the alphabet $\Sigma = \{0\} \cup \mathcal{G} \cup \{[,], (,), \cdot, +, \otimes\}$. Strings over Σ which do not encode valid formulas are deemed to represent the trivial tensor formula 0 of order 1×1.

Let F be a tensor formula of order $m \times n$. Its *size*, denoted $|F|$, is $\max\{m, n\}$ and its *length* $L(F)$ is the number of symbols in its string representation. It is easy to show that $|F| \leq 2^{O(L(F))}$. The upper bound is attained when F is an iterated tensor product.

We also consider two restrictions on tensor formulas. A formula F is called *tame*, if for each of its subformulas F', $|F'| \leq 2 \cdot m_F$, where m_F denotes the maximal size of F's *atomic* subformulas. Moreover, F is called *simple* if it does not contain the tensor product. Simple tensors are tame, since the only size-increasing operation is omitted.

The inductive definition of a tensor formula F associates with F a binary parse tree. The root of this tree subtends F, and each other node subtends a unique subformula of F. By a *collection of subformulas of F*, we formally mean a set of nodes in the parse tree of F. Observe that two subformulas F_1 and F_2 occurring as distinct subtrees within the parse tree of F are distinct regardless of what the individual string encodings of F_1 and F_2 might be.

Proposition 3. 1. *Testing whether a string encodes a valid tensor formula and if so, computing its order, can be done in deterministic polytime.*

2. *Testing the validity and computing the order of tame tensor formulas can be performed by a deterministic auxiliary pushdown automaton in logspace and polytime.*

3. *Testing the validity and computing the order of simple formulas can be done in deterministic logspace.*

Proof. (1) Let M be the Turing machine which, on an input string w, rejects and halts if the bracketing or operator structure of w are illegal. This can be tested in logspace. If w is legal, then M continues by running the function *order* described by the following pseudo-code:

function *order* (tensor F) : (nat, nat);
var k, ℓ, m, n : nat;
begin case F **in**:
  atomic: determine order of F and store it in (k,ℓ);
    return (k,ℓ);
  $(G+H)$: $(k,\ell) :=$ *order*(G); $(m,n) :=$ *order*(H);
    if $k \neq m$ **or** $\ell \neq n$ **then** halt and reject **fi**;
    return (k,ℓ);
  $(G \cdot H)$: $(k,\ell) :=$ *order*(G); $(m,n) :=$ *order*(H);
    if $\ell \neq m$ **then** halt and reject **fi**;
    return (k,n);
  $(G \otimes H)$: $(k,\ell) :=$ *order*(G); $(m,n) :=$ *order*(H);
    return $(k\ell, mn)$;
  esac;
end.

The *order* function may be implemented on M, using a tape in a pushdown like fashion to handle the recursive calls. Hence M operates in polytime, since M performs a depth-first search of the formula, and since polynomial space is sufficient to keep track of the orders in binary notation. The initial call *order*(F) thus returns the order of F.

(2) We proceed as in (1) but before calling *order* we determine m_F, the maximal size of F's atomic subformulas, and during the recursive calls, we control that for each visited subformula G we find $|G| \leq 2 \cdot m_F$. If this is not the case, then M halts and rejects. Since $O(\log |F|)$ bits now suffice to compute and store orders of subformulas, the complete algorithm may be implemented on a deterministic auxiliary logspace and polytime bounded pushdown automaton.

(3) Although simple formulas don't contain Kronecker products, a direct implementation of the algorithm described in (2) without the case $G \otimes H$ would still lead to an algorithm running on an auxiliary pushdown automaton. But we can do better. The main difficulty is to check that the tensor operands in a sum have the same order. Indeed observe that if no sum is involved, i.e., if the input formula F only involves nodes of the form $G \cdot H$, then a depth-first search of F can check that the orders of the tensor subformulas involved match properly. This can be done by a logspace-bounded deterministic Turing machine. If the input is an arbitrary simple formula F, i.e., a formula containing $+$-nodes and $\cdot$-nodes, then the following strategy, first explained informally, applies. Each leaf F_i in the parse tree of F will be considered in turn. Pick the unique path p from F_i to the root F. Then, at each $\cdot$-node G along p, attach just enough of the subtree rooted at G to be able to deduce the order of the subformula G (any included $\cdot$-node forces the inclusion of both its children, while choosing left descendants whenever a choice arises; thus only the left child of any included

+-node outside p gets included). There results a unique "F_i-subtree," having a unique order (if it has an order at all) computable in logspace just as if the F_i-subtree only contained $\cdot$-nodes. An induction shows that F is valid and has order $k \times \ell$ if and only if each leaf F_i of F yields a F_i-subtree of order $k \times \ell$.

To formalize this strategy and to justify its logspace implementation, fix an atomic subformula F_1 of the input. Then let F be an *arbitrary* subformula of the input—hence F does not necessarily subtend F_1. Now define the collection $C_{F_1}(F)$ of subformulas of F recursively as follows (leaving out the F_1 subscript momentarily):

1. if F is an atomic formula, then $C(F) = \{F\}$.
2. if $F = (G + H)$, then
$$C(F) = \begin{cases} C(G) \cup \{F\} & \text{if } F_1 \in C(G) \text{ or } F_1 \notin C(G) \cup C(H), \\ C(H) \cup \{F\} & \text{if } F_1 \in C(H). \end{cases}$$
3. if $F = (G \cdot H)$, then $C(F) = C(G) \cup C(H) \cup \{F\}$.

When F *is* the input formula, the collection $C_{F_1}(F)$ defines the unique "F_1-subtree" alluded to in our earlier informal description. But now, it should be clear that a logspace Turing machine can test whether a subformula belongs to $C_{F_1}(F)$. Such a test can guide a depth-first search of $C_{F_1}(F)$. Since $C_{F_1}(F)$ includes precisely one child of each +-node and both children of each $\cdot$-node, it is indeed easy to test validity and to compute the order of $C_{F_1}(F)$, by treating it as a formula with $\cdot$-nodes alone. Now combining the validity tests and the order computations of $C_{F_i}(F)$ for each atomic formula F_i results in an algorithm to test validity and to compute the order of F. □

Definition 4. *For each semiring $\mathcal{S}$ and each k and each ℓ we define $\mathrm{val}_{\mathcal{S}}^{k,\ell} : \mathbb{T}_{\mathcal{S}}^{k,\ell} \to \mathbb{M}_{\mathcal{S}}^{k,\ell}$, that is, we associate with each tensor formula F of order $k \times \ell$ its $k \times \ell$ matrix "value," as follows:*

$$\mathrm{val}_{\mathcal{S}}^{k,\ell}(F) = \begin{cases} F & \textit{if } F \textit{ is atomic} \\ \mathrm{val}_{\mathcal{S}}^{k,\ell}(G) + \mathrm{val}_{\mathcal{S}}^{k,\ell}(H) & \textit{if } F = (G + H) \\ \mathrm{val}_{\mathcal{S}}^{k,m}(G) \cdot \mathrm{val}_{\mathcal{S}}^{m,\ell}(H) & \textit{if } F = (G \cdot H) \textit{ and } G \in \mathbb{T}_{\mathcal{S}}^{k,m} \\ \mathrm{val}_{\mathcal{S}}^{k/m,\ell/n}(G) \otimes \mathrm{val}_{\mathcal{S}}^{m,n}(H) & \textit{if } F = (G \otimes H) \textit{ and } H \in \mathbb{T}_{\mathcal{S}}^{m,n}. \end{cases}$$

The corresponding mappings t-$\mathrm{val}_{\mathcal{S}}^{k,\ell}$ *and* s-$\mathrm{val}_{\mathcal{S}}^{k,\ell}$ *are defined by restricting the domain of* $\mathrm{val}_{\mathcal{S}}^{k,\ell}$ *to the set of all tame and simple tensor formulas over $\mathcal{S}$, respectively. Tensor formulas of order 1×1 are called* scalar tensor formulas, *and we simply write* $\mathrm{val}_{\mathcal{S}}$ *for the* $\mathrm{val}_{\mathcal{S}}^{1,1}$ *function. We do the same in case of* t-$\mathrm{val}_{\mathcal{S}}^{1,1}$ *and* s-$\mathrm{val}_{\mathcal{S}}^{1,1}$ *by omitting the superscripts.*

The non-zero tensor problem is defined as follows:

Definition 5. *Let $\mathcal{S}$ be a semiring. The* non-zero tensor formula problem *over semiring $\mathcal{S}$ is the set $0 \neq \mathrm{val}_{\mathcal{S}}$ of all scalar tensor formulas F for which $\mathrm{val}_{\mathcal{S}}(F) \neq 0$. The corresponding evaluation problems for tame and simple tensors are defined in the analogous way, and are written $0 \neq \text{t-val}_{\mathcal{S}}$ and $0 \neq \text{s-val}_{\mathcal{S}}$, respectively.*

The next proposition shows that considering scalar tensor formulas for the non-zero problem is no restriction at all.

Proposition 6. *The non-zero tensor formula problem $0 \neq \mathrm{val}_{\mathcal{S}}$ is polytime many-one equivalent to the* general non-zero tensor formula problem *which is defined as follows:*

- *Given a tensor formula F over $\mathcal{S}$ of order $k \times \ell$ and two integers i and j expressed in binary, such that $(i, j) \in [k] \times [\ell]$, is $(\mathrm{val}_{\mathcal{S}}^{k,\ell}(F))_{i,j} \neq 0$?*

The corresponding evaluation problems for tame and simple tensor formulas are logspace many-one equivalent to the general problem if restricted accordingly.

Proof. Since the non-zero tensor problem $0 \neq \mathrm{val}_{\mathcal{S}}$ trivially reduces by $F \mapsto (F, 1, 1)$ to an instance of the general problem, it remains to show that the latter reduces to $0 \neq \mathrm{val}_{\mathcal{S}}$. Let (F, i, j) be an instance of the general non-zero tensor problem and assume $I(F) = [k] \times [\ell]$. First we check with the help of the algorithm described in Proposition 3 whether $(i, j) \in [k] \times [\ell]$, and if so proceed as follows.

In case F is tame or simple we output

$$F_{i,j} = \left(e_i^k\right) \cdot F \cdot \left(e_j^\ell\right)^{\mathsf{T}},$$

where e_i^m is the ith unit row vector of length m. Obviously, the formula $F_{i,j}$ satisfies $(\mathrm{val}_{\mathcal{S}}^{k,\ell}(F))_{i,j} = \mathrm{val}_{\mathcal{S}}(F_{i,j})$, and it is easily seen that it is logspace constructible from F. If F is an unrestricted tensor formula the vectors e_i^k and e_j^ℓ may be of exponential length. Hence, we can not directly write $F_{i,j}$. To overcome this, note that k can be expressed in polytime as $k = m_1 \cdot m_2 \cdots m_t$, where each m_r is the row dimension of some atomic subformula of F. Expressibility of k in this way is readily verified by induction on F. But then, for $1 \leq i \leq k$,

$$e_i^k = e_{i_1}^{m_1} \otimes e_{i_2}^{m_2} \otimes \cdots \otimes e_{i_t}^{m_t},$$

where i_r, for $1 \leq r \leq t$, is defined by

$$i_r = 1 + \left\lfloor \frac{i - 1 - \sum_{j=1}^{r} (i_{j-1} - 1) \cdot M_j}{M_{r+1}} \right\rfloor,$$

for $M_r = m_r \cdot m_{r+1} \cdots m_t$ and $i_0 = M_{t+1} = 1$.

An analogous formula expresses e_j^ℓ using the column dimensions of atomic subformulas of F. $\square$

3 A Tensor Formula for the Permanent

In this section we show how to use the copying feature of the Kronecker operation to "parallelize" certain computations in a controlled way. A first direct application of Lemma 7 will be the construction of a tensor formula for the permanent of a matrix A. This resembles Berkowitz' theorem [10] giving a simple tensor formula for the determinant. Further applications are given in the next section.

To understand the statement of Lemma 7 below, keep in mind a situation in which it is required to compute, say $(A+B)(C+D+E)(F)$, where A, B, C, D, E and F are $k \times k$ matrices. Lemma 7 describes a preliminary step which uses tensors to produce a large block matrix having the expansion products ACF, ADF, AEF, BCF, BDF, and BEF as diagonal blocks. This particular application of Lemma 7 would require the parameters $n = 3$, $m_1 = 2$, $m_2 = 3$, and $m_3 = 1$.

Lemma 7. *Let $m_1, \ldots, m_n$ be natural numbers, and write $N = \prod_{i=1}^{n} m_i$. Let a sequence $\mathbf{A} = (A_{i,j_i})$, with $1 \le i \le n$ and $1 \le j_i \le m_i$, of $k \times k$ matrices over a semiring be given. Consider the $k \times k$ matrix*

$$\prod_{i=1}^{n} (A_{i,1} + \cdots + A_{i,m_i}) = \sum_{\substack{1 \le j_1 \le m_1 \\ \vdots \\ 1 \le j_n \le m_n}} A_{1,j_1} A_{2,j_2} \ldots A_{n,j_n}. \tag{1}$$

There is a polytime Turing machine which computes, on input $\mathbf{A}$, a tensor formula $F_n(\mathbf{A})$ evaluating to a $kN \times kN$ matrix having each $k \times k$ summand $A_{1,j_1} A_{2,j_2} \ldots A_{n,j_n}$ occurring in Equation (1) as a $k \times k$ block along its diagonal, and zero elsewhere.

Proof. The statement is proved by induction on n. Obviously,

$$F_1(\mathbf{A}) = \sum_{j=1}^{m_1} \left(D_j^{m_1} \otimes A_{1,j} \right)$$

has the required form for $n = 1$, where D_i^m is $m \times m$ "dot matrix" having one in position (i, i) and zeros elsewhere. Then, for $n > 1$ assume that we already have a matrix $F_{n-1}(\mathbf{A})$ of the appropriate form representing the summands of $\prod_{i=1}^{n-1} (A_{i,1} + \cdots + A_{i,m_i})$. In order to obtain $F_n(\mathbf{A})$ we have yet to multiply by the factor $(A_{n,1} + \cdots + A_{n,m_n})$. This is done recursively as follows:

$$F_n(\mathbf{A}) = \left(I_{m_n} \otimes F_{n-1}(\mathbf{A}) \right) \cdot \sum_{j=1}^{m_n} \left(\left(D_j^{m_n} \otimes \bigotimes_{i=1}^{n-1} I_{m_i} \right) \otimes A_{n,j} \right),$$

which satisfies our requirements. $\square$

As a first application we construct a tensor formula for the permanent of an $n \times n$ matrix A, which is defined by

$$\mathrm{perm}(A) = \sum_{\sigma \in \mathrm{S}_n} \prod_{i=1}^{n} a_{i\sigma(i)} = \sum_{\sigma \in \mathrm{S}_n} a_{1\sigma(1)} \cdot a_{2\sigma(2)} \cdots a_{n\sigma(n)},$$

where S_n denotes the symmetric group on n elements.

Recall, that a function h is polytime many-one reducible to a function g if there is a mapping f such that for all x, $h(x) = g(f(x))$.

Theorem 8. *There is a polytime computable function f which, given an $n \times n$ matrix A over a semiring $\mathcal{S}$, computes a scalar tensor formula $f(A)$ such that $\mathrm{val}_{\mathcal{S}}(f(A)) = \mathrm{perm}(A)$.*

Proof. Write P_σ for the $n \times n$ permutation matrix associated with σ from the symmetric group S_n, writing $T_{i,j}$ instead of P_σ when σ is the transposition (i,j) and setting each $T_{i,i}$ to I_n. First, we use Lemma 7 to construct a tensor formula F_n evaluating to a square matrix of order $n! \cdot n^n$, which has each of the $n!$ blocks $P_\sigma^{\otimes n}$ of size n^n on its diagonal, and zeros elsewhere. It is well known that each P_σ is uniquely expressible as $T_{1,j_1} \cdot T_{2,j_2} \cdots T_{n,j_n}$ for some $i \le j_i \le n$, in which case Proposition 1 implies $P_\sigma^{\otimes n} = T_{1,j_1}^{\otimes n} \cdot T_{2,j_2}^{\otimes n} \cdots T_{n,j_n}^{\otimes n}$. Thus, applying Lemma 7 to the sequence $\mathbf{T} = (T_{i,j_i}^{\otimes n})$ with $1 \le i \le n$ and $i \le j_i \le n$ yields the desired tensor formula $F_n = F_n(\mathbf{T})$.

Next, let $M(A) = ((\bigotimes_{i=1}^{n} I_i) \otimes A^{\otimes n}) \cdot F_n$. This matrix has the $n!$ blocks $(AP_\sigma)^{\otimes n}$ of size n^n along its diagonal, and zeros elsewhere. Note that the diagonal elements of AP_σ are $a_{1\sigma(1)}, a_{2\sigma(2)}, \ldots, a_{n\sigma(n)}$. Some diagonal entry in $(AP_\sigma)^{\otimes n}$ therefore is the coveted element $\prod_{i=1}^{n} a_{i\sigma(i)}$. A crucial observation is that the relative position of this entry within $(AP_\sigma)^{\otimes n}$ does not depend on σ, and that this relative position is precisely the relative position of the single non-zero entry in another matrix, namely $\bigotimes_{i=1}^{n} D_i^n$. Pre-multiplying and post-multiplying $M(A)$ by the symmetric matrix $(\bigotimes_{i=1}^{n} I_i) \otimes (\bigotimes_{i=1}^{n} D_i^n)$ therefore results in a square matrix of order $n! \cdot n^n$ having zeros everywhere, except at $n!$ diagonal positions holding the values $\prod_{i=1}^{n} a_{i\sigma(i)}$ for each $\sigma \in \mathrm{S}_n$.

It remains to sum up the diagonal entries into a scalar tensor. This is done by pre- and post-multiplying further by $(1)_{n! \cdot n^n}$ and its transpose, respectively, where $(1)_k$ denotes the all-ones row vector of length k. The partial product to the left of $M(A)$, namely

$$\left((1)_{n! \cdot n^n} \right) \cdot \left(\bigotimes_{i=1}^{n} I_i \otimes \bigotimes_{i=1}^{n} D_i^n \right), \tag{2}$$

in fact simplifies to $(1)_{n!} \otimes (\bigotimes_{i=1}^{n} e_i^n)$, where e_i^n is the ith length-n unit row vector. Since the partial product to the right of $M(A)$ is the transpose of (2), and since $(1)_{n!} = \bigotimes_{i=1}^{n} (1)_i$, the complete formula reads as

$$\mathrm{perm}(A) = \left((1)_{n!} \otimes \bigotimes_{i=1}^{n} e_i^n \right) \cdot \left(\left(\bigotimes_{i=1}^{n} I_i \otimes A^{\otimes n} \right) \cdot F_n \right) \cdot \left((1)_{n!} \otimes \bigotimes_{i=1}^{n} e_i^n \right)^{\mathsf{T}}.$$

This completes the construction. $\square$

Corollary 9. *There is a polytime computable function f which, given an $n \times n$ matrix A over $\mathbb{N}$, computes a scalar tensor formula $f(A)$ such that* $\mathrm{val}_{\mathbb{N}}(f(A)) = \mathrm{perm}(A)$. $\square$

Observe that a tensor formula for the determinant would result from inserting a factor -1 before every non-identity permutation T_{i,j_i} in the above proof, i.e., applying Lemma 7 to $\mathbf{T} = ((-1)^{1+\delta_{ij_i}} \cdot T_{i,j_i}^{\otimes n})$.

Since computing the permanent of $0/1$-matrices is #P-complete under polytime Turing reduction [38] and $\mathbb{N}$ is embedded in every infinite field, we conclude:

Corollary 10. *For* any *infinite field $\mathbb{F}$,* $\mathrm{val}_{\mathbb{F}}$ *is* #P*-hard under polytime Turing reductions.* $\square$

4 Algebraic Turing Machines

We introduce an algebraic computation model which is very useful for proving upper bounds for the evaluation problem of unrestricted, tame, and simple tensor formulas.

Arithmetic or algebraic models have received much attention in computer science during the last decade. In [9] Beimel and Gál introduced algebraic branching programs—a very natural algebraic generalization of nondeterministic branching programs. In this section we define algebraic Turing machines in a similar way. The difference with ordinary nondeterministic Turing machines is that now every transition has a weight taken from a semiring $\mathcal{S}$. The weight of a computation path then is defined as the product of the weights of the transitions taken by the Turing machine along that path. An algebraic Turing machine has two uses, namely computing a function $f_M : \Sigma^* \to \mathcal{S}$ and accepting a language L_M. For $w \in \Sigma^*$, the function f_M is defined as the sum of the weights of the accepting computations of M on input w (where the product and the sum are taken in $\mathcal{S}$). Obviously, this is a generalization of a #P Turing machine, which counts the number of accepting paths. The language L_M is defined as $\{ w \in \Sigma^* \mid f_M(w) \neq 0 \}$. More formally, the definition of an algebraic Turing machine reads as follows:

Definition 11. *An* algebraic Turing machine over a semiring $\mathcal{S}$ *is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, where Q, $\Sigma \subseteq \Gamma$, Γ, $q_0 \in Q$, $B \in \Gamma$, and $F \subseteq Q$ are defined as for ordinary Turing machines and δ is the* transition relation *taking the form*

$$\delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{L, S, R\} \times \mathcal{S}.$$

Let the configurations, *the* next move relation $\vdash_M$, *and its reflexive transitive closure $\vdash_M^*$ be defined as for ordinary Turing machines. A move from configuration C to C', i.e., $C \vdash_M C'$, using the transition $(p, a, q, b, m, s) \in \delta$, gets assigned a* weight s. *In this way, a weight may be associated any pair of* *configurations* (C, C')*, where we define the weight to be* 0 *whenever* C' *is not reachable from* C *in a single transition.*

The weight of a particular computation $C_1 \vdash_M \cdots \vdash_M C_t$ *of* M *is the* product *of the weights of the successive moves in the computation. For completeness, we define the weight of the computation to be* 1 *if* $t = 1$*. On input* w *machine* M *computes the value* $f_M(w)$*, which is defined as the* sum *of the weights of all accepting computations* $C_0(w) \vdash_M^* C_t(w)$*, where* $C_0(w)$ *is the* initial configuration *on input* w *and* $C_t(w)$ *is a final configuration.*

The above definition is adapted in a straightforward way to define algebraic polytime Turing machines, algebraic logspace bounded Turing machines, and algebraic auxiliary space (and time) bounded pushdown automata. Note that Allender *et al.* [2] have introduced an auxiliary pushdown model similar to ours, called the generalized LOGCFL machine.

Definition 12. *1. We define the* generalized counting class $\mathcal{S}$-#P *as the set of all functions* $f : \Sigma^* \to \mathcal{S}$ *such that there is a a polytime algebraic Turing machine* M *over* $\mathcal{S}$ *which computes* f*.*

2. The generalized language class $\mathcal{S}$-P *is the set of all languages* $L \subseteq \Sigma^*$ *such that there is a polytime Turing machine* M *over* $\mathcal{S}$ *for which* $w \in L$ *if and only if* $f_M(w) \neq 0$*.*

The corresponding generalized counting and languages classes for algebraic auxiliary logspace polytime bounded pushdown automata and algebraic logspace bounded Turing machines are defined in the obvious way, and are denoted by $\mathcal{S}$-#LOGCFL*,* $\mathcal{S}$-#L *and* $\mathcal{S}$-LOGCFL*,* $\mathcal{S}$-L*, respectively.*

We state the following special cases without proof:

Proposition 13. *1.* $\mathbb{B}$-P $=$ NP.

2. $\mathbb{Z}_q$-P $=$ MOD$_q$-P.

3. $\mathbb{F}_2$-P $= \oplus$P.

4. $\mathbb{N}$-#P $=$ #P *and* $\mathbb{N}$-P $=$ NP.

5. $\mathbb{Z}$-#P $=$ *Gap*P. □

Statements similar to Proposition 13 also hold for the generalized counting and languages classes obtained from algebraic auxiliary logspace polytime bounded pushdown automata and logspace Turing machines.

5 The Complexity of Tensor Evaluation Problems

This section contains completeness results on the computational complexity of the tensor formula evaluation problem over certain semirings. We need some preliminaries.

We require the notion of a certificate of a tensor formula index. Intuitively, a certificate is to a tensor formula entry what a proof tree is to a Boolean formula. Certificates and their weights will be defined in such a way that the entry at index $(i,j) \in [k] \times [\ell]$ in $\mathrm{val}_{\mathcal{S}}^{k,\ell}(F)$ is equal to the sum of the weights of all the certificates of F at (i,j). We now make this precise.

Given a collection $C = \{F_1, \dots, F_t\}$ of subformulas of a formula F, a *weight function* for C is a mapping w that assigns to each F_i an element of $\mathcal{S}$, and an *entry selector* for C is a mapping ε that assigns to each F_i a unique index $\varepsilon(F_i) = (k,\ell) \in I(F_i)$. We say that "$\varepsilon$ selects the entry (k,ℓ) from F_i." Let C_1 and C_2 be disjoint collections. Given the entry selectors ε_1 for C_1 and ε_2 for C_2, we simply write $\varepsilon = \varepsilon_1 \cup \varepsilon_2$ to specify the entry selector $\varepsilon : (C_1 \cup C_2) \to \mathcal{S}$ with $\varepsilon(F) := \varepsilon_1(F)$ if $F \in C_1$ and $\varepsilon(F) := \varepsilon_2(F)$ if $F \in C_2$.

Definition 14. *Let F be a tensor formula over a semiring $\mathcal{S}$ and let $(i,j) \in I(F)$. A* certificate *for (F,i,j) is a pair (C,ε), where C is a collection of subformulas of F* weighted *by a function w and ε is an entry selector for C, subject to the following inductive conditions:*

1. *If F is an atomic formula, then $C = \{F\}$, entry selector $\varepsilon(F) = (i,j)$, and $w(F) = (F)_{i,j}$, where $(F)_{i,j}$ is the (i,j)th entry of matrix F.*
2. *If $F = (G + H)$ and (C_G, ε_G) is a certificate for (G,i,j) and (C_H, ε_H) is a certificate for (H,i,j), then either*
   (a) *$C = C_G \cup \{F\}$, $\varepsilon = \varepsilon_G \cup \{F \mapsto (i,j)\}$, and $w(F) = w(G)$ or*
   (b) *$C = C_H \cup \{F\}$, $\varepsilon = \varepsilon_H \cup \{F \mapsto (i,j)\}$, and $w(F) = w(H)$.*
3. *If $F = (G \cdot H)$ and (C_G, ε_G) is a certificate for (G,i,k) and (C_H, ε_H) is a certificate for (H,k,j) for some k, then $C = C_G \cup C_H \cup \{F\}$, $\varepsilon = \varepsilon_G \cup \varepsilon_H \cup \{F \mapsto (i,j)\}$, and $w(F) = w(G) \cdot w(H)$.*
4. *If $F = (G \otimes H)$ with $I(G) = [p] \times [q]$, (C_G, ε_G) is a certificate for (G,k,ℓ) and (C_H, ε_H) is a certificate for (H,m,n), $i = p \cdot (k-1) + m$, and $j = q \cdot (\ell - 1) + n$, then $C = C_G \cup C_H \cup \{F\}$, the entry selector $\varepsilon = \varepsilon_G \cup \varepsilon_H \cup \{F \mapsto (i,j)\}$, and the weight $w(F) = w(G) \cdot w(H)$.*

The weight *$w(C,\varepsilon)$ of this certificate (C,ε) for (F,i,j) is defined as $w(F)$. Finally, for a tensor formula F and $(i,j) \in I(F)$ let $\mathrm{Cert}(F,i,j)$ denote the* set of all certificates for (F,i,j). *For scalar tensor formulas we simply write* $\mathrm{Cert}(F)$ *instead of* $\mathrm{Cert}(F,1,1)$.

By induction one obtains:

Lemma 15. *Let F be a scalar tensor formula over a semiring $\mathcal{S}$. Then*

$$\mathrm{val}_{\mathcal{S}}(F) = \sum_{(C,\varepsilon) \in \mathrm{Cert}(F)} w(C,\varepsilon).$$

Proof. Let F be a tensor formula of order $k \times \ell$. We prove the more general statement

$$(\mathrm{val}_{\mathcal{S}}^{k,\ell}(F))_{i,j} = \sum_{(C,\varepsilon)\in \mathrm{Cert}(F,i,j)} w(C,\varepsilon) \tag{3}$$

for each $(i,j) \in I(F)$. Recall that $(\mathrm{val}_{\mathcal{S}}(F))_{i,j}$ is the (i,j)th entry of the matrix $\mathrm{val}_{\mathcal{S}}(F)$.

If F is atomic, there is a unique certificate $(\{F\},\varepsilon)$ with $\varepsilon(F) = (i,j)$ for (F,i,j) having weight $(\mathrm{val}_{\mathcal{S}}^{k,\ell}(F))_{i,j}$. Hence (3) holds. Now, assume that (3) holds for formulas G and H of order less than F, at all indices. We distinguish three cases:

1. If $F \in \mathbb{T}_{\mathcal{S}}^{k,\ell}$ and $F = (G+H)$, then for each $(i,j) \in I(F)$ by definition and induction hypothesis,

$$\begin{aligned}
(\mathrm{val}_{\mathcal{S}}^{k,\ell}(F))_{i,j} &= (\mathrm{val}_{\mathcal{S}}^{k,\ell}(G+H))_{i,j} \\
&= (\mathrm{val}_{\mathcal{S}}^{k,\ell}(G))_{i,j} + (\mathrm{val}_{\mathcal{S}}^{k,\ell}(H))_{i,j} \\
&= \sum_{(C,\varepsilon)\in \mathrm{Cert}(G,i,j)} w(C,\varepsilon) + \sum_{(C,\varepsilon)\in \mathrm{Cert}(H,i,j)} w(C,\varepsilon) \\
&= \sum_{(C,\varepsilon)\in \mathrm{Cert}(G+H,i,j)} w(C,\varepsilon) \\
&= \sum_{(C,\varepsilon)\in \mathrm{Cert}(F,i,j)} w(C,\varepsilon),
\end{aligned}$$

   because a certificate for (F,i,j) ensures that either subformula G or H is selected, but not both. Thus, from a certificate for (G,i,j) or (H,i,j) it is obvious how to construct one for (F,i,j), and the weights are preserved by definition.

2. If $F \in \mathbb{T}_{\mathcal{S}}^{k,m}$, $F = (G \cdot H)$, and $G \in \mathbb{T}_{\mathcal{S}}^{k,\ell}$, then for each $(i,j) \in I(F)$ by definition and induction hypothesis,

$$\begin{aligned}
&= (\mathrm{val}_{\mathcal{S}}^{k,m}(G \cdot H))_{i,j} \\
&= \sum_{n=1}^{\ell} (\mathrm{val}_{\mathcal{S}}^{k,\ell}(G))_{i,n} \cdot (\mathrm{val}_{\mathcal{S}}^{\ell,m}(H))_{n,j} \\
&= \sum_{n=1}^{\ell} \left(\sum_{(C,\varepsilon)\in \mathrm{Cert}(G,i,n)} w(C,\varepsilon) \right) \cdot \left(\sum_{(C,\varepsilon)\in \mathrm{Cert}(H,n,j)} w(C,\varepsilon) \right) \\
&= \sum_{(C,\varepsilon)\in \mathrm{Cert}(G\cdot H,i,j)} w(C,\varepsilon) \\
&= \sum_{(C,\varepsilon)\in \mathrm{Cert}(F,i,j)} w(C,\varepsilon),
\end{aligned}$$

where the next to last equality follows because each of the certificates "created at node $G \cdot H$" by the local choices $1, 2, \ldots, \ell$ gets assigned as weight the product of the weights of the children of the node at the entries selected by this local choice.

3. The remaining case $F = (G \otimes H)$ is treated analogously.

This proves (3), and in particular the case of scalar tensor formulas. $\square$

Next we prove upper bounds on the complexity of tensor evaluation. For the unrestricted, tame, and simple tensor evaluation problem we obtain the following upper bounds:

Lemma 16. *For any finitely generated semiring $\mathcal{S}$,*

1. $\mathrm{val}_{\mathcal{S}} \in \mathcal{S}\text{-}\#\mathrm{P}$,
2. $\text{t-val}_{\mathcal{S}} \in \mathcal{S}\text{-}\#\mathrm{LOGCFL}$, *and*
3. $\text{s-val}_{\mathcal{S}} \in \mathcal{S}\text{-}\#\mathrm{L}$.

Proof. (1) Let M be the following Turing machine: given a scalar tensor formula F, the machine M guesses a string and verifies that it encodes a certificate for F. If it is a certificate, the weight of this particular computation equals the weight of the certificate. In the other case M rejects. More precisely, M runs a recursive procedure *verify*, on a tensor formula F and $(i, j) \in I(F)$, checking the requirements of Definition 14. All transition weights in M are assigned 1, if not explicitly specified. The pseudo-code of *verify* reads as follows:

```
procedure verify (tensor F, nat i, nat j);
var k, ℓ, m, n, p, q : nat;
begin if (i,j) ∉ I(F) then halt and reject fi;
      case F in:
      atomic:   read (i,j)th entry of F;
                take transition having weight equal to that value;
      (G + H):  nondeterministically guess either
                   call verify(G,i,j) or call verify(H,i,j);
      (G · H):  (p,q):=order(G);
                guess index 1 ≤ k ≤ q;
                call verify(G,i,k); call verify(H,k,j);
      (G ⊗ H):  (p,q):=order(G);
                deterministically compute indices (k,ℓ) and (m,n)
                   such that i = k · (p − 1) + m and j = ℓ · (q − 1) + n;
                call verify(G,k,ℓ); call verify(H,m,n);
      esac;
end.
```

Obviously, M guesses a certificate (C, ε) during a computation and the weight of this computation equals $w(C, \varepsilon)$. The procedure *verify* may be implemented on M by using a tape in a pushdown like fashion to handle the recursive calls to *verify*, operating in polytime because it performs a depth-first search of the formula. Hence, calling *verify*$(F, 1, 1)$ on input F gives the desired result $\mathrm{val}_{\mathcal{S}}(F)$ by Lemma 15.

(2) First we check whether the input is tame. This can be done on a deterministic auxiliary pushdown automaton in logspace and polytime by Proposition 3. Then we proceed as in (1), now implementing the *verify* procedure on an algebraic auxiliary pushdown automaton. Since on tame input F, $O(\log|F|)$ bits are sufficient to store orders of subformulas and thus to control the recursive calls in the computation, *verify* actually runs on a logspace and polytime bounded algebraic auxiliary pushdown automaton.

(3) Observe that the pushdown is only necessary to handle the recursive calls to *verify* in the case of tensor products. This is because the logspace depth-first search algorithm used in Proposition 3 to compute the order of a sum-free formula can be adapted to mimic the recursive calls *verify*(G, i, k) and *verify*(H, k, j). Since a simple formula doesn't contain tensor products, *verify* may actually be implemented on an algebraic Turing machine running in logspace. □

5.1 Unrestricted Tensor Formulas

We prove completeness properties of the evaluation problem for unrestricted tensor formulas. At first we complete our considerations, started in Section 3, on tensor formulas over the natural numbers. An immediate consequence of Theorem 8, Lemma 16, and Valiant's result [38] is:

Theorem 17. $\mathrm{val}_{\mathbb{N}}$ *is* #P*-complete under polytime Turing-reductions.* □

Obviously, this also implies that evaluating tensor formulas over the Booleans or over $\mathbb{Z}_q$ are Turing-complete for NP and MOD_q-P, respectively, but in both cases we can do better as stated in the next theorem.

Theorem 18. *The following problems are complete under polytime many-one reductions:*

1. $0 \neq \mathrm{val}_{\mathbb{B}}$ *is* NP*-complete.*
2. $0 \neq \mathrm{val}_{\mathbb{Z}_q}$ *is* MOD_q-P*-complete.*

Proof. The containment in NP and MOD_q-P follows from Lemma 16 and Proposition 13. For the hardness we use reductions from 3SAT and from MOD_q-3SAT. We start with the Boolean case.

Let $f = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ be a Boolean formula with m clauses and n variables in 3CNF, i.e., each clause C_i, for $1 \leq i \leq m$, is the disjunction of three variables x_j or negated variables $\bar{x}_j$, with $1 \leq j \leq n$. Let $a = (a_1, \ldots, a_n) \in \{0,1\}^n$ be an assignment and define $d_a = (a_1, \ldots, a_n, \bar{a}_1, \ldots, \bar{a}_n)$ as its encoding. We encode C_i by a length $2n$ column vector c_i as follows:

$$c_i = \sum_{x_j \text{ is in } C_i} \left(e_1^2 \otimes e_j^n\right) + \sum_{\bar{x}_j \text{ is in } C_i} \left(e_2^2 \otimes e_j^n\right),$$

where e_i^n is the ith unit row vector of length n. Note that

$$d_a \cdot \left(e_i^2 \otimes e_j^n\right)^\mathsf{T} = \begin{cases} a_j & \text{if } i = 1, \\ \bar{a}_j & \text{otherwise.} \end{cases}$$

Obviously, clause C_i is satisfied by a if and only if $d_a \cdot {c_i}^\mathsf{T} = 1$. Moreover, formula f is satisfied by a if and only if a satisfies all clauses C_i, for $1 \le i \le m$, if and only if

$$d_a^{\otimes m} \cdot \left(\bigotimes_{i=1}^{m} c_i\right)^\mathsf{T} = \bigotimes_{i=1}^{m} \left(d_a \cdot {c_i}^\mathsf{T}\right) = \bigotimes_{i=1}^{m} 1 = 1. \tag{4}$$

Obviously, every encoding d_a of an assignment $a \in \{0,1\}^n$ can be generated starting from $(e_1^2 \otimes (1)_n)$, where $(1)_n$ is the all-ones row vector of length n, by appropriately applying transposition $(i, n+i)$ or the identity for $1 \le i \le n$. Let $T_{i,n+i}$ be the $2n \times 2n$ matrix realizing transposition $(i, n+i)$ for $1 \le i \le n$. Now we proceed as in the construction of the tensor formula for the permanent, applying Lemma 7 to the sequence $\mathbf{A} = (A_{i,j_i})$, with $1 \le i \le n$ and $1 \le j_i \le 2$, where $A_{i,1} = I_{2n}^{\otimes m}$ and $A_{i,2} = T_{i,n+i}^{\otimes m}$. This results in a tensor formula $F_n = F_n(\mathbf{A})$ evaluating to a square matrix of order $2^n \cdot (2n)^m$ with 2^n blocks of size $(2n)^m$ along the main diagonal and zeros elsewhere, such that the non-zero blocks are the mth tensor powers of $2n \times 2n$ permutation matrices $A_{1,j_1} \cdot A_{2,j_2} \cdots A_{n,j_n}$ for some $1 \le j_i \le 2$. Let M be equal to

$$\left(I_2^{\otimes n} \otimes \left(e_1^2 \otimes (1)_n\right)^{\otimes m}\right) \cdot F_n = \begin{pmatrix} d_{a_1}^{\otimes m} & & & 0 \\ & d_{a_2}^{\otimes m} & & \\ & & \ddots & \\ 0 & & & d_{a_{2^n}}^{\otimes m} \end{pmatrix}, \tag{5}$$

where $a_i, a_j \in \{0,1\}^n$ and $a_i \ne a_j$ if $i \ne j$. This matrix is of order $2^n \times 2^n \cdot (2n)^m$, it contains the mth power of every possible boolean assignment to n variables, in a brick like fashion, from its upper left corner to its lower right corner, and it vanishes everywhere else. Thus, by Equations (4) and (5),

$$M \cdot \left(I_2^{\otimes n} \otimes \left(\bigotimes_{i=1}^{m} c_i\right)^\mathsf{T}\right) = \begin{pmatrix} f(a_1) & & & 0 \\ & f(a_2) & & \\ & & \ddots & \\ 0 & & & f(a_{2^n}) \end{pmatrix}.$$

To decide 3SAT it remains to sum up these entries by pre- and post multiplying by the appropriate vector $(1)_{2^n}$ and its transpose, respectively. Since $(1)_{2^n}$ equals $(1)_2^{\otimes n}$ this results in

$$\left((1)_2^{\otimes n}\right) \cdot \left(I_2^{\otimes n} \otimes \left(e_1^2 \otimes (1)_n\right)^{\otimes m}\right) \cdot F_n \cdot \left(I_2^{\otimes n} \otimes \left(\bigotimes_{i=1}^{m} c_i\right)^\mathsf{T}\right) \cdot \left((1)_2^{\otimes n}\right)^\mathsf{T},$$

which equals $\sum_{a\in\{0,1\}^n} f(a)$. Finally, observe that the resulting tensor formula is deterministic polytime computable from f. This completes the construction in the Boolean case.

The difficulty in the MOD$_q$-3SAT reduction to $0\neq\mathrm{val}_{\mathbb{Z}_q}$ is that when $\mathcal{S}=\mathbb{Z}_q$ it is *no longer true* that "(C_i is satisfied by a) $\Rightarrow$ ($d_a\cdot {c_i}^{\mathsf{T}}=1\in\mathcal{S}$)" and that "($C_i$ is not satisfied by a) $\Rightarrow$ ($d_a\cdot {c_i}^{\mathsf{T}}=0\in\mathcal{S}$)." This held in the Boolean case because the sum $x+y+z$ in $\mathbb{B}$ is precisely $x\vee y\vee z$. But summing $\{0,1\}$-truth values in this way over $\mathbb{Z}_q$ no longer works. To get around this problem, instead of $x+y+z$, we employ the polynomial

$$x^3+y^3+z^3+(q-1)(x^2y+x^2z+y^2z)+xyz \mod q$$

as the basis for the encoding of a clause $x\vee y\vee z$. Observe that when $x,y,z\in\{0,1\}$, this polynomial evaluates to $0\in\mathbb{Z}_q$ if $x=y=z=0$ and it evaluates to $1\in\mathbb{Z}_q$ otherwise. This suggests encoding a Boolean assignment a as $(d_a)^{\otimes 3}$, and defining the encoding c_i of a clause C_i in such a way that $(d_a)^{\otimes 3}\cdot {c_i}^{\mathsf{T}}$ yields the result of evaluating the above polynomial on the truth values x,y and z induced by a on the literals of C_i. In more detail, suppose that C is the clause $(x_i\vee\bar{x}_j\vee x_k)$. Then the encoding of C (or of any 3-SAT clause for that matter) is the sum of seven terms, arising from the seven monomials in the above polynomial. For example, the summand arising from the monomial $(q-1)x_i^2\bar{x}_j$ is the $(q-1)$-fold iterated sum of $(e_1^2\otimes e_i^n)\otimes(e_1^2\otimes e_i^n)\otimes(e_2^2\otimes e_j^n)$. This should suffice to clarify the idea. The rest of the MOD$_q$-3SAT proof follows with minor changes to the lines of the proof in the Boolean case. $\square$

Corollary 19. $0\neq\mathrm{val}_{\mathbb{F}_2}$ *is* $\oplus$P*-complete under polytime many-one reduction.* $\square$

In the remainder of this subsection we consider the scalar tensor evaluation problem over the integers.

Theorem 20. $\mathrm{val}_{\mathbb{Z}}$ *is GapP-complete under polytime many-one reductions.*

Proof. By Lemma 16, $\mathrm{val}_{\mathbb{Z}}\in\mathbb{Z}$-#P, and the latter equals *Gap*P by Proposition 13. For the hardness, let $f\in$ *Gap*P. Then f is the difference of two #P functions, so by Proposition 13, $f=f_{M_1}-f_{M_2}$ where M_1 and M_2 are algebraic polytime Turing machine over $\mathbb{N}$. Now observe that #-3SAT reduces to $\mathrm{val}_{\mathbb{Z}}$ using a construction similar to that in the proof of Theorem 18, employing the polynomial

$$x^3+y^3+z^3+(-1)(x^2y+x^2z+y^2z)+xyz$$

as the basis for the coding of a clause $x\vee y\vee z$. Thus, f_{M_1} and f_{M_2} polytime many-one reduce to evaluating scalar tensor formulas F_1 and F_2 over $\mathbb{Z}$, respectively, and therefore f reduces to evaluating the scalar tensor formula $F_1+((-1)\cdot F_2)$ over $\mathbb{Z}$. $\square$

5.2 Restricted Tensor Formulas

We consider the $0 \neq \mathrm{val}_{\mathcal{S}}$ problem for tame and simple tensor formulas. For proving hardness we use a characterization of $\mathcal{S}$-#LOGCFL in terms of logspace uniform sequences of *semi-unbounded circuits over* $\mathcal{S}$ of polynomial size and logarithmic depth. Such circuits are similar to semi-unbounded AND-OR-circuits except that AND- and OR-gates are replaced by product and sum gates, respectively(hence the sums have unbounded fan-in). Note that negations are allowed only at input gates. The uniformity notion carries over to this type of circuits: a circuit sequence $C = (C_n)$ is called uniform, if there is a logspace bounded deterministic Turing machine that given an input of length n produces a description of the nth circuit in the sequence. We prove the following characterization of $\mathcal{S}$-#LOGCFL in terms of arithmetic semi-unbounded circuits. Note that Allender *et al.* [2] proved a similar result for height bounded auxiliary pushdown automata.

Theorem 21. *Let $\mathcal{S}$ be any finitely generated commutative semiring. The class of functions computed by logspace polytime algebraic auxiliary pushdown automata over $\mathcal{S}$ is equal to the class of functions computed by logspace uniform polynomial size and logarithmic depth semi-unbounded circuits over $\mathcal{S}$.*

Proof. In the case of the Boolean semiring and the naturals the relationship between semi-unbounded circuits and auxiliary pushdown machines was elucidated by Venkatesvaran [40] and Vinay [41]. Obviously, the simulation of a semi-unbounded circuit by an auxiliary pushdown automaton extends to the general setting of arbitrary finitely generated (not necessarily commutative) semirings—at $+$-gates guess one of the children, and at $\times$-gates proceed with the left child followed by the right one. For the other way around, we use a result of Niedermeier and Rossmanith [30]. They showed how to decompose a computation path represented by realizable pairs of surface configurations of an auxiliary pushdown automaton in a unique way to construct a semi-unbounded circuit which preserves the number of accepting computation paths. Following the lines of their proof it is easy to see that this simulation is still valid for finitely generated *commutative* semirings. □

Now our hardness result reads as follows:

Lemma 22. *Let (C_n) be a logspace-computable sequence of semi-unbounded logarithmic depth* algebraic *circuits over a finitely generated commutative semiring $\mathcal{S}$. There is a logspace computable function f that on a length n input computes a tame scalar tensor formula F_n such that for each $x = (x_1, \ldots, x_n) \in \{0,1\}^n$,*

$$C_n(x) = \mathrm{val}_{\mathcal{S}}(F_n \cdot d_x),$$

where $d_x = (x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n)$.

Proof. By a deterministic logspace algorithm C_n can be transformed into the following normal form circuit of depth $2d$:

1. The gates are partitioned into levels $V^0 \cup \cdots \cup V^{2d}$. Inputs to gates in level V^{j+1} are in level V^j. Here V^0 is the set of input gates of C_n.

2. Gates in levels V^{2j} are $+$-gates, denoted by letters h, gates in levels V^{2j-1} are $\times$-gates, denoted by letters g.

3. For some $m > 0$ let $|V^{2j}| = 2m$ for $j = 0, \ldots, d$ and $|V^{2j-1}| = m$ for $j = 1, \ldots, d$.

4. $+$-gates have out-degree 1.

Further we assume that the gates are numbered within the levels in such a way that for $j = 1, \ldots, d$ and $i = 1, \ldots, m$ the gate $g_i^{2j-1} \in V^{2j-1}$ has inputs $h_i^{2j-2}, h_{i+m}^{2j-2} \in V^{2j-2}$ (see Figure 1). This is possible by item 4. Now we fix

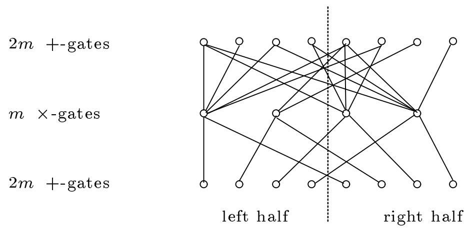

Figure 1: Three consecutive levels in the circuit (values propagate upward).

an input assignment $x \in \{0,1\}^n$ and denote the values computed at the gates simply by the gate names. Let $L^{2j} = (h_1^{2j}, \ldots, h_m^{2j})$, $R^{2j} = (h_{m+1}^{2j}, \ldots, h_{2m}^{2j})$, and $H^{2j} = (L^{2j}, R^{2j})$. Observe that $L^{2j} = H^{2j} \cdot L^\top$ and $R^{2j} = H^{2j} \cdot R^\top$, where $L = (I_m, 0_m)$ and $R = (0_m, I_m)$. Here I_m and 0_m, resp., are the identity and the all-zero matrix of order $m \times m$, respectively.

Clearly, $h_i^0 = x_\ell$ or $h_i^0 = \bar{x}_\ell$ according to whether h_i^0 is labeled with x_ℓ or $\bar{x}_\ell$. Hence, $(h_1^0, \ldots, h_{2m}^0) = P \cdot d_x$ for an appropriate 0/1-matrix P.

Now, consider some gate h_i^{2j} with inputs $g_{\ell_1}^{2j-1}, \ldots, g_{\ell_r}^{2j-1}$. Then

$$h_i^{2j} = \sum_{\tau \in \{\ell_1, \ldots, \ell_r\}} g_\tau^{2j-1} = \sum_{\tau \in \{\ell_1, \ldots, \ell_r\}} h_\tau^{2j-1} \cdot h_{\tau+m}^{2j-1}.$$

Denote by T_i^{2j} the $(m \times m)$ diagonal matrix $\sum_{\tau \in \{\ell_1, \ldots, \ell_r\}} D_\tau^m$, where D_i^m is the $m \times m$ dot-matrix having one in position (i,i) and zeros elsewhere. The above equation can be expressed as

$$\begin{aligned} h_i^{2j} &= L^{2j-2} \cdot T_i^{2j} \cdot (R^{2j-2})^\top \\ &= (H^{2j-2} \cdot L^\top) \cdot T_i^{2j} \cdot (H^{2j-2} \cdot R^\top)^\top. \end{aligned}$$

The whole level H^{2j} can be expressed as

$$H^{2j} = H^{2j-2} \cdot (L^\top \cdot T^{2j} \cdot R) \cdot ((1)_{2m} \otimes (H^{2j-2})^\top),$$

where T^{2j} is the order $m \times 2m^2$ matrix $(T_1^{2j}, \ldots, T_{2m}^{2j})$ and $(1)_{2m}$ is the all-ones row vector of length $2m$.

After d iterations the output level is represented as a tensor formula F_n. Since the D^{2j} depend only on the gate interconnections, the construction of F_n is not harder than that of C_n. Formula F_n is tame, since sizes of subformulas never exceeds $2m^2$. □

By Lemma 16 and 22 we obtain:

Corollary 23. t-val$_{\mathcal{S}}$ *is* $\mathcal{S}$-#LOGCFL-*complete under logspace many-one reduction for any finitely generated commutative semiring* $\mathcal{S}$. □

In case of naturals and integers we conclude:

Corollary 24. *The following functions are complete under logspace many-one reductions:*

1. t-val$_{\mathbb{N}}$ *is* #LOGCFL-*complete.*
2. t-val$_{\mathbb{Z}}$ *is* *Gap*LOGCFL-*complete.* □

In particular for the non-zero tensor problem we find:

Corollary 25. *The following problems are complete under logspace many-one reductions:*

1. $0 \neq$ t-val$_{\mathbb{B}}$ *is* LOGCFL-*complete.*
2. $0 \neq$ t-val$_{\mathbb{Z}_q}$ *is* MOD$_q$-LOGCFL-*complete.*
3. $0 \neq$ t-val$_{\mathbb{F}_2}$ *is* $\oplus$LOGCFL-*complete.* □

In the remainder of this section we consider simple tensor formulas. As a complement to Lemma 16, we state the following without proof:

Theorem 26. s-val$_{\mathcal{S}}$ *is* $\mathcal{S}$-#L-*complete under logspace many-one reductions for any finitely generated semiring.* □

In the case of naturals and integers we conclude:

Corollary 27. *The following functions are complete under logspace many-one reductions:*

1. s-val$_{\mathbb{N}}$ *is* #L-*complete.*
2. s-val$_{\mathbb{Z}}$ *is* *Gap*L-*complete.* □

As an immediate consequence of Theorem 26 we conclude:

Corollary 28. *The following problems are complete under logspace many-one reductions:*

1. $0 \neq$ s-val$_{\mathbb{B}}$ *is* NL-*complete.*

2. $0 \neq \text{s-val}_{\mathbb{Z}_q}$ *is* MOD_q-L-*complete.*

3. $0 \neq \text{s-val}_{\mathbb{F}_2}$ *is* $\oplus$L-*complete.* □

Proof. We only sketch the proof. It is known that computing iterated matrix products over $\mathbb{B}$ and $\mathbb{Z}_q$ is hard for NL and $\mathrm{MOD}_q L$, respectively (see [15, 24]). Hence, the above problems are hard for the classes mentioned, as they include in particular computing iterated matrix products. On the other hand, $0 \neq \text{s-val}_{\mathbb{B}} \in$ NL and $0 \neq \text{s-val}_{\mathbb{Z}_q} \in \oplus$L are easy to see. □

6 An Application: Non-uniform Simulations

In this section we prove that the Boolean variants of the tensor formula evaluation problems non-uniformly reduce to their counterparts over $\mathbb{F}_2$. For the proof of the main theorem we need the following well-known result due to Zippel [44] and Schwartz [32] and some basic facts on algebraic Turing machine and auxiliary pushdown automata.

Lemma 29. *Let K be a field. Let $p \in K[z_1, \ldots, z_t]$ be a polynomial of total degree D and let $S \subseteq K$ be a finite set. If $r_1, \ldots, r_t$ are chosen independently and uniformly at random from S, then*

$$\Pr[\, p(r_1, \ldots, r_t) = 0 \mid p \neq \mathbf{0}\,] \leq \frac{D}{|S|},$$

where $\mathbf{0}$ is the zero polynomial.

Concerning algebraic Turing machines basics, we show how to efficiently simulate algebraic machines over $\mathbb{F}_{2^d}$ by machines over the smaller semiring $\mathbb{F}_2$. The proof follows the lines of Beimel and Gál [9], where a similar result was shown for arithmetic branching programs. We start with an easy lemma (where the meaning of the OR is as expected, i.e., it is the Boolean OR on $\{0,1\}$ although the latter come from $\mathbb{F}_2$):

Lemma 30. *Let d be a natural number. If $f_{M_i} \in \mathbb{F}_2$-#P ($f_{M_i} \in \mathbb{F}_2$-#LOGCFL, $f_{M_i} \in \mathbb{F}_2$-#L, respectively) for $0 \leq i < d$, then*

$$f = \bigvee_{i=0}^{d-1} f_{M_i}$$

belongs to $\mathbb{F}_2$-#P ($\mathbb{F}_2$-#LOGCFL, $\mathbb{F}_2$-#L, respectively). The description of an algebraic machine M computing f can be constructed by a logspace Turing machine given $M_0, \ldots, M_{d-1}$.

Proof. The OR of all f_{M_i}'s equals $1 + \prod_{i=0}^{d-1}(1 + f_{M_i}) \pmod 2$. In order to obtain M first introduce in every M_i a new final state which is reachable by a single weight 1 nondeterministic step from the initial state, hence computing $1 + f_{M_i}$. Let M_i' be this new machine. The product $\prod_{i=0}^{d}(1 + f_{M_i})$ is computed by connecting the M'_i in sequence, i.e., if M'_i reaches a final state, then the computation is continued with M'_{i+1} for $0 \leq i < d$. Finally, it remains to add 1 to obtain the desired result, which is done as above. Observe that the machine M constructed in this way is logspace computable from the M_i's, and that the function computed by M belongs to the same class as that of the f_{M_i}'s. $\square$

The next lemma gives the simulation of an algebraic Turing machine over $\mathbb{F}_{2^d}$ by an equivalent machine over $\mathbb{F}_2$.

Lemma 31. *Let function $f_M \in \mathbb{F}_{2^d}$-#P ($f_M \in \mathbb{F}_{2^d}$-#LOGCFL, $f_M \in \mathbb{F}_{2^d}$-#L, respectively). Then there is a $f_{M'} \in \mathbb{F}_2$-#P ($f_{M'} \in \mathbb{F}_2$-#LOGCFL, $f_{M'} \in \mathbb{F}_2$-#L, respectively) for which*

$$f_M(w) \neq 0 \quad \textit{if and only if} \quad f_{M'}(w) \neq 0.$$

There is a logspace Turing machine which computes, given the algebraic machine M, a description of machine M' over $\mathbb{F}_2$.

Proof. Recall, that $\mathbb{F}_{2^d} = \mathbb{F}_2[x]/p(x)$, where $\mathbb{F}_2[x]$ denotes the ring of polynomials in x with coefficients from $\mathbb{F}_2$, the polynomial $p(x) \in \mathbb{F}_2[x]$ is irreducible over $\mathbb{F}_2$ and of degree d, and $\mathbb{F}_2[x]/p(x)$ denotes the ring of polynomials in x of degree at most $d-1$, where addition and multiplication are of polynomials modulo the polynomial $p(x)$.

We simulate $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ with an algebraic Turing machine over $\mathbb{F}_2$ exploiting the fact that the weights are polynomials in x of degree at most $d-1$. First we construct d algebraic Turing machines $M_0, M_1, \ldots, M_{d-1}$ over $\mathbb{F}_2$. To obtain M_i, each state of M is duplicated d times, where the ith copy of $q \in Q$ is denoted by (q, i). The state (q, i), for $0 \leq i < d$, is "responsible" for the coefficient of x^i. The initial state is $(q_0, 0)$ and the final states of M_i are all (q, i), where $q \in F$. Let $(q, a, q', b, m, p'(x)) \in \delta$ be a transition of M, where $q, q' \in Q$, $a, b \in \Gamma$, $m \in \{L, S, R\}$, and $p'(x) \in \mathbb{F}_2[x]/p(x)$. For every $0 \leq j < d$ let

$$\sum_{k=0}^{d} a_{j,k} \cdot x^k = p'(x) \cdot x^j \pmod{p(x)}.$$

In M_i we add the d^2 transitions $((q, j), a, (q', k), b, m, a_{j,k})$ for $0 \leq j, k < d$ with weight $a_{j,k} \in \mathbb{F}_2$.

By induction, for every input w, the sum of the weights of accepting computations starting in $(q_0, 0)$ to state (q, j) in M_i is equal to the coefficient of x^j in the sum of weights of the accepting computations starting from q_0 and leading to q in M. On input w the Turing machine M computes 0, i.e., is equal to the zero polynomial, if and only if for every $0 \leq i < d$ the sum of all accepting computations on input w in M_i over $\mathbb{F}_2$ is zero. Thus, $f_M(w) \neq 0$ if and only if $f_{M_i}(w) \neq 0$ for some i, with $0 \leq i < d$.

It remains to compute the OR of all the M_i's, which is done with Lemma 30, resulting in a polytime algebraic machine M'. By the discussion above, $f_M(w) \neq 0$ if and only if $f_{M'}(w) \neq 0$. Observe that M' is logspace computable from M.

Obviously, the proof carries over to algebraic auxiliary logspace polytime bounded pushdown automata and logspace bounded Turing machines, respectively. □

Now we are ready to state the main theorem of this section. As usual, the nonuniform versions of complexity classes are denoted with a trailing /*poly*. For further details we refer to [25].

Theorem 32. *1.* NP/*poly* $\subseteq$ $\oplus$P/*poly,*

2. LOGCFL/*poly* $\subseteq$ $\oplus$LOGCFL/*poly, and*

3. NL/*poly* $\subseteq$ $\oplus$L/*poly.*

Proof. Consider $L \in \mathrm{NP}/poly$. By Theorem 18, an input w together with its polynomial-length advice can be transformed in polytime into a scalar tensor formula F over $\mathbb{B}$, of length $n = poly(|w|)$, such that

$$\mathrm{val}_{\mathbb{B}}(F) = 1 \quad \text{if and only if} \quad w \in L.$$

Let $s = \lceil \log M(F) \rceil$, where $M(F) = \max\{\, |G|^2 \mid G \text{ is a subformula of } F\,\}$. Clearly, $s \leq 2n$ if F has length n. Hence, for any subformula G there exists an s-bit binary encoding of the set $I(G)$. For fixed G let $bin_\ell(i,j)$ denote the ℓth bit of the string encoding $(i,j) \in I(G)$.

For each subformula G consider a set $\{z_G^1, \dots, z_G^s\}$ of variables. Let

$$m_{(G,i,j)}(z) := \prod_{bin_\ell(i,j)=1} z_G^\ell.$$

For a certificate (C,ε) of F let $M_{(C,\varepsilon)}(z) = \prod_{G \in S} m_{(G,\varepsilon(G))}$. Finally, let

$$p_F(z) := \sum_{\substack{(C,\varepsilon)\in \mathrm{Cert}(F)\\ w(C,\varepsilon)\neq 0}} M_{(C,\varepsilon)}(z).$$

Let $t \leq 2n^2$ be the total number of variables. Obviously, $\deg p_F(z) \leq t$. The binary encoding ensures that monomials in p_F cannot cancel each other out. Thus, $\mathrm{val}_{\mathbb{B}}(F) = 1$ if and only if $p_F(z) \neq \mathbf{0}$ (the zero polynomial) by Lemma 15. Consider a field $K = \mathbb{F}_{2^d}$ such that $|K| > 2c \cdot n^2$, where c is the cardinality of the input alphabet Σ. From Lemma 29 we conclude that

$$\mathrm{val}_{\mathbb{B}}(F) = 1 \quad \text{implies} \quad \Pr[p_F(r) = 0] < 1/c$$

and

$$\mathrm{val}_{\mathbb{B}}(F) = 0 \quad \text{implies} \quad \Pr[p_F(r) = 0] = 1,$$

for $r = (r_1, \dots, r_t)$ chosen uniformly at random from K^t. For n independent trials $r^1, \dots, r^n$ we obtain

$$\Pr[p_F(r^1) = \cdots = p_F(r^n) = 0] < 1/c^n,$$

if $\mathrm{val}_{\mathbb{B}}(F) = 1$. As there are at most c^n tensor formulas of length n a well-known counting argument (see, e.g., [1]) shows that there are assignments $r^1, \ldots, r^n$ such that for *every* tensor formula F of length n,

$$\mathrm{val}_{\mathbb{B}}(F) = 0 \quad \text{if and only if} \quad p_F(r^1) = \cdots = p_F(r^n) = 0,$$

where $p_F(z) \in K[z_1, \ldots, z_t]$ is the polynomial assigned to F.

The key observation is that $p_F(r) = \mathrm{val}_K(F_r)$ for each $r \in K^t$, where $F_r = (F, \omega_r)$ is a *weighted version* of F, obtained from F by multiplying entries (i,j) of any subformula G of F by the value $\omega_r(G,i,j) := m_{(G,i,j)}(r)$. Similar constructions were used in [9, 17].

To be precise, the value of a *weighted tensor formula* (F, ω) over a semiring $\mathcal{S}$, where $\omega : \bigcup_{G \in S} G \times I(G) \to \mathcal{S}$ is a *weight function* and S is the collection of all subformulas of F, is defined by recursively setting $(\mathrm{val}_{\mathcal{S}}^{k,\ell}(F,\omega))_{i,j}$ equal to:

$$\begin{cases} \omega(F,i,j) \cdot (\mathrm{val}_{\mathcal{S}}^{k,\ell}(F))_{i,j} & \text{if } F \text{ is atomic} \\ \omega(F,i,j) \cdot (\mathrm{val}_{\mathcal{S}}^{k,\ell}(G,\omega) + \mathrm{val}_{\mathcal{S}}^{k,\ell}(H,\omega))_{i,j} & \text{if } F = (G+H) \\ \omega(F,i,j) \cdot (\mathrm{val}_{\mathcal{S}}^{k,m}(G,\omega) \cdot \mathrm{val}_{\mathcal{S}}^{m,\ell}(H,\omega))_{i,j} & \text{if } F = (G \cdot H) \text{ and } G \in \mathbb{T}_{\mathcal{S}}^{k,m} \\ \omega(F,i,j) \cdot (\mathrm{val}_{\mathcal{S}}^{k/m,\ell/n}(G,\omega) \otimes \mathrm{val}_{\mathcal{S}}^{m,n}(H,\omega))_{i,j} & \text{if } F = (G \otimes H) \text{ and } H \in \mathbb{T}_{\mathcal{S}}^{m,n}. \end{cases}$$

In particular, for the weight function ω_r it obviously follows that $p_F(r) = \mathrm{val}_K(F, \omega_r)$ for each $r \in K^t$. Moreover, $\mathrm{val}_K(F, \omega_r)$ is in K-#P, by altering the procedure *verify* in the proof of Lemma 16 according to whether the assignment r is fixed or given as part of the input. Thus, we are left with n algebraic polytime Turing machines $M_1, \ldots, M_n$ over $K = \mathbb{F}_{2^d}$ computing $\mathrm{val}_K(F, \omega_{r_1}), \ldots, \mathrm{val}_K(F, \omega_{r_n})$, respectively. Then by Lemma 31 there are algebraic polytime Turing machines $M'_1, \ldots, M'_n$ over $\mathbb{F}_2$ such that $f_{M_i}(w) \neq 0$ if and only if $f_{M'_i}(w) \neq 0$ for some i, with $1 \leq i \leq n$. The OR of the $f_{M'_i}$ is done with Lemma 30 resulting in an algebraic Turing machine M' over $\mathbb{F}_2$.

Now above we have shown the existence of an algebraic polytime Turing machine M' over $\mathbb{F}_2$ having the property that

$$\mathrm{val}_{\mathbb{B}}(F) = 1 \quad \text{if and only if} \quad f_{M'}(F\#\alpha) = 1,$$

where α is the further advice $r^1, \ldots, r^n$, of $n \cdot t \cdot d = O(n^3 \log n)$ bits. Since $\mathbb{F}_2$-P $= \oplus$P by Proposition 13, it follows that $L \in \oplus\mathrm{P}/\mathit{poly}$.

In the case $L \in \mathrm{LOGCFL}/\mathit{poly}$ ($L \in \mathrm{NL}/\mathit{poly}$, respectively), the intermediate logspace-computable scalar tensor formula F over $\mathbb{B}$ is tame, by Corollary 25 (simple by Corollary 28, respectively). The algebraic machine M' over $\mathbb{F}_2$ operating on $F\#\alpha$ is then an algebraic logspace polytime auxiliary pushdown automaton (an algebraic logspace Turing machine, respectively), so we conclude by analogs of Proposition 13 that $L \in \oplus\mathrm{LOGCFL}/\mathit{poly}$ ($L \in \oplus\mathrm{L}/\mathit{poly}$, respectively). □

Since $0 \neq \mathrm{val}_{\mathbb{F}_2}$ is $\oplus$P-hard (Corollary 19), the proof of Theorem 32 in fact exhibits a non-uniform polytime reduction (as defined in [43]) from $0 \neq \mathrm{val}_{\mathbb{B}}$ to

$0 \neq \mathrm{val}_{\mathbb{F}_2}$. Similarly, it yields a non-uniform logspace reduction from $0 \neq \text{t-val}_{\mathbb{B}}$ to $0 \neq \text{t-val}_{\mathbb{F}_2}$, and from $0 \neq \text{s-val}_{\mathbb{B}}$ to $0 \neq \text{s-val}_{\mathbb{F}_2}$.

Note further that in the case of tame tensor formulas, the construction used in proving Theorem 32 requires only $O(n^2 \log^2 n)$ random bits, while the construction in [20] uses $O(n^3)$ random bits.

7 Conclusions

A gratifying overall picture emerges from the present work. The three most important nondeterministic complexity classes NL, LOGCFL, and NP were long known to have complete problems of an algebraic flavour, and the link between LOGCFL and unbounded fan-in Boolean circuits was known to generalize. But here these classes are captured by a single problem, Boolean tensor formula evaluation. This problem, together with the notion of an (auxiliary pushdown) algebraic Turing machine, unifies the treatment of the three classes beyond expectation. In particular, inclusions of NL, LOGCFL, and NP in their parity counterparts become instances of one and the same phenomenon. Counting the number of nondeterministic computations also generalizes smoothly: evaluating tensor formulas over the natural numbers yields the usual counting classes, and evaluating them over other semirings yields the analog concept for more general algebraic Turing machines.

The completeness results of the last section can be generalized to tensor formulas over residue class rings in a straightforward way. Moreover, standard techniques as surveyed, e.g., in [8], show that evaluation problems for (general, tame, simple) tensor formulas over finite fields $\mathbb{F}_{p^d}$, for prime p, are computationally equivalent to those over $\mathbb{F}_p$. Indeed, a field element of $\mathbb{F}_{p^d}$ corresponds to a "d-vector" over $\mathbb{F}_p$. The idea is similar to that used in Lemma 31 to represent computations of an algebraic Turing machine over $\mathbb{F}_{p^d}$ by d computations of algebraic Turing machines over $\mathbb{F}_p$, each responsible for one component of the formula. The $\mathbb{F}_{p^d}$-outcome is non-zero if and only if the outcome of the first OR the second OR ... OR the dth $\mathbb{F}_p$-computation is non-zero. By cascading $\mathbb{F}_p$-computations $d-1$ times the outcomes are forced to be either 1 or 0 (by Fermat's Little Theorem). This, eventually, can be exploited to simulate the big OR condition.

In this paper we have only considered *formulas* and shown that these capture nondeterministic complexity classes. But what happens if we replace formulas with *circuits*, that is, if we allow reusing the output of a subevaluation more than once? Then the complexity, even in the Boolean case, seems to rise sharply because circuits can produce intermediate tensors of double-exponential size. The evaluation problem for such circuits can be solved in nondeterministic exponential time. On the other hand, the large intermediate tensors created are heavily contrived, so that proving hardness seems challenging.

Let us finally mention an interesting further aspect of our completeness results. In linear algebra there is an intimate relationship (in the algebraic as well as in the computational sense) between computing rank, matrix formulas, or determinants [10, 28]. No similar relations are known in the world of tensors. Even worse, in multi-linear algebra no single concept is known to exist as unifying and as versatile as the determinant in linear algebra (see [22]). The results of Section 5 in particular say that the complexity of tensor evaluation problems over fields decisively depends on the characteristic of the field (unless some of the classes coincide or some inclusions hold). On the other hand, Håstad [23] has shown that computing the *rank of a tensor* is at least as hard as NP *regardless of the field's characteristic*. Tensor rank, which is a central notion in algebraic complexity theory (see [12]), is a natural generalization of matrix rank. However, while matrix rank can be expressed efficiently in matrix calculus [28], Håstad's result together with the characterizations from Section 5 tells us that tensor rank *cannot* be expressed efficiently in tensor calculus.

References

[1] L. Adleman. Two theorems on random polynomial time. In *Proceedings of the 19th Foundations of Computer Science*, pages 75–83. IEEE Computer Society Press, 1978.

[2] E. Allender, J. Jiao, M. Mahajan, and V. Vinay. Non-commutative arithmetic circuits: depth reduction and size lower bounds. *Theoretical Computer Science*, 209:47–86, 1998.

[3] E. Allender and M. Ogihara. Relationships among PL, #L, and the determinant. *RAIRO—Theoretical Informatics and Applications*, 30:1–21, 1996.

[4] C. Àlvarez and B. Jenner. A very hard log space counting class. *Theoretical Computer Science*, 107:3–30, 1993.

[5] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I.* Texts in Theoretical Computer Science. Springer Verlag, 1995.

[6] D. A. Mix Barrington. Bounded-width polynomial size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

[7] F. Bédard, F. Lemieux, and P. McKenzie. Extensions to Barrington's M-program model. *Theoretical Computer Science*, 107:31–61, 1993.

[8] R. Beigel. The polynomial method in circuit complexity. In *Proceedings of the 8th Structure in Complexity Theory*, pages 82–95. IEEE Computer Society Press, 1993.

[9] A. Beimel and A. Gál. On arithmetic branching programs. *Journal on Computer and Systems Sciences*, 59(2):195–220, 1999.

[10] S. J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Information Processing Letters*, 18:147–150, 1984.

[11] G. Buntrock, C. Damm, U. Hertrampf, and C. Meinel. Structure and importance of logspace MOD-classes. *Mathematical Systems Theory*, 25:223–237, 1992.

[12] P. Burgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic Complexity Theory*, volume 315 of *Grundlehren Der Mathematischen Wissenschaften*. Springer Verlag, 1997.

[13] S. R. Buss. The Boolean formula value problem is in ALOGTIME. In *Proceedings of the 19th Symposium on Theory of Computing*, pages 123–131. ACM Press, 1987.

[14] S.R. Buss, S. Cook, A. Gupta, and V. Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM Journal on Computing*, 21(4):755–780, 1992.

[15] C. Damm. Problems complete for $\oplus$L. *Information Processing Letters*, 36:247–250, 1990.

[16] C. Damm. DET = $L^{\#L}$? Technical Report 8, Fachbereich Informatik der Humboldt-Universität zu Berlin, 1991.

[17] C. Damm. Depth-efficient simulation of boolean semi-unbounded circuits by arithmetic ones. *Information Processing Letters*, 69:175–179, 1999.

[18] J. A. Eisele and R. M. Mason. *Applied Matrix and Tensor Analysis*. Wiley & Sons Publisher, 1970.

[19] S. Fenner, L. Fortnow, and S. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48:116–148, 1994.

[20] A. Gál. Semi-unbounded fan-in circuits: Boolean vs. arithmetic. In *Proceedings of the 10th Structure in Complexity*, pages 82–87, 1995.

[21] A. Gál and A. Wigderson. Boolean complexity classes vs. their arithmetic analogs. *Random Structures and Algorithms*, 9(1-2):99–111, 1996.

[22] I. M. Gelfand, M. M. Kapranov, and A. V. Celevinskij. *Discriminants, resultants, and multidimensional determinants*. Mathematics: Theory & Applications. Birkhäuser Verlag, 1994.

[23] J. Håstad. Tensor rank is NP-complete. *Journal of Algorithms*, 11:644–654, 1990.

[24] N. Immernan and S. Landau. The complexity of iterated multiplication. *Information and Computation*, 116:103–116, 1995.

[25] R. Karp and R. Lipton. Turing machines that take advice. *L'enseignement mathématique*, 28:191–209, 1982.

[26] W. Kuich and A. Salomaa. *Semirings, Automata, Languages*, volume 5 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, 1986.

[27] Ch. Meinel. Polynomial size Ω-branching programs and their computational power. *Information and Computation*, 85(2):163–182, 1990.

[28] K. Mulmuley. A fast parallel algorithm to compute the rank of a matrix over an arbitrary field. In *Proceedings of 18th Symposium on Theory of Computing*, pages 338–339, ACM Press, 1986.

[29] K. Mulmuley, U. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7:105–113, 1987.

[30] R. Niedermeier and P. Rossmanith. Unambiguous auxiliary pushdown automata and semi-unbounded fan-in circuits. *Information and Computation*, 118(2):227–245, May 1995.

[31] C. H. Papadimitriou and S. Zachos. Two remarks on the power of counting. In *Proceedings of 6th Conference on Theoretical Computer Science*, number 145 of *LNCS*, pages 269–275. Springer Verlag, 1983.

[32] J. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the Association for Computing Machinery*, 27:701–717, 1980.

[33] W.-H. Steeb. *Kronecker Product of Matrices and Applications*. Wissenschaftsverlag, 1991.

[34] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Proceedings of the 5th ACM Symposium on the Theory of Computing*, pages 1–9. ACM Press, 1973.

[35] V. Strassen. The asymptotic spectrum of tensors. *Journal für die reine und angewandte Mathematik*, 384:102–152, 1988.

[36] S. Toda. *Computational Complexity of Counting Complexity Classes*. PhD thesis, Tokyo Institute of Technology, Department of Computer Science, Tokyo, 1991.

[37] S. Toda. PP is as hard as the polynomial time hierarchy. *SIAM Journal on Computing*, 20:865–877, 1991.

[38] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[39] L. G. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.

[40] H. Venkateswaran. Properties that characterize LOGCFL. *Journal of Computer and System Sciences*, 43:380–404, 1991.

[41] V. Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In *Proceedings of the 6th Structure in Complexity Theory*, pages 270–284. IEEE Computer Society Press, 1991.

[42] I. Wegener. *The Complexity of Boolean Functions*. Wiley-Teubner series in computer science. B. G. Teubner & John Wiley, 1987.

[43] A. Wigderson. $NL/poly \subseteq \oplus L/poly$. In *Proceedings of the 9th Structure in Complexity Theory*, pages 59–62. IEEE Computer Society Press, 1994.

[44] R. Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the Symposium on Symbolic and Algebraic Manipulation*, number 72 of *LNCS*, pages 216–226. Springer Verlag, 1979.

A Some Comments on the Connection Between Tensors and Matrices

Formally defined, a tensor t over $\mathcal{S}$ is a mapping $t : [d_1] \times \cdots \times [d_k] \to \mathcal{S}$, where $d_1, \ldots, d_k \in \{1, 2, \ldots\}$ and $[d_i] := \{1, \ldots, d_i\}$ for each i. The shorthand notation for t is $(t_{i_1 \ldots i_t})$. We allow for also upper indices, like in $t = (t_{i_1 \ldots i_r}^{j_1 \cdots j_s})$. We call such a tensor an (r, s)*-tensor*. To avoid double indices we write tensors without specifying the number of indices and their ranges, like in $(t_{i \ldots j}^{k \cdots \ell})$. To define the tensor operations we further simplify and consider as operands only $(2, 2)$ tensors. This is enough to see how the exact general definition would be.

Let $t = (t_{ij}^{k\ell})$ and $s = (s_{mn}^{op})$ be tensors. We consider the following three operations: the *tensor product* $t \otimes s$ is defined as $(u_{ijmn}^{k\ell op}) = (t_{ij}^{k\ell} \cdot s_{mn}^{op})$. If corresponding indices of t and s vary in the same range, their *sum* $s + t$ is defined, which is $(u_{ij}^{k\ell}) = (t_{ij}^{k\ell} + s_{ij}^{k\ell})$. Finally, if for tensor t the ranges of one lower index and one upper index, say i and ℓ, coincide, t's *junction*, also called *partial trace, via i and ℓ* exists and is defined by $\Gamma_i^\ell t = (\sum_\mu t_{\mu j}^{k\mu})$, where the sum ranges over all μ in the joint range of i and ℓ.

Similar to tensor formulas (formally based on matrix operations) we can define *tensor expressions* and their evaluation on base of the just defined tensor operations. Tameness and simplicity conditions can be introduced as in the matrix terminology and similar to the non-zero tensor formula problem we consider the *non-zero tensor expression problem*: decide whether a well-formed tensor expression describes a scalar non-zero tensor (an $(1, 1)$-tensor with a single entry). It turns out, that both problems are computationally equivalent, i.e., can efficiently be transformed into each other. We argue informally about that only in the general case, referring to polytime computations. The tame or simple case are similar—the additional conditions ensure that transformations can be computed within logspace.

A matrix is the special case of a $(1, 1)$-tensor. The definition of the sum is the same for matrices and tensors. The tensor product of matrices is not readily defined as a matrix, since this would be a $(2, 2)$-tensor. However, by arranging it's entries in a block-wise fashion, we get the definition of a tensor product for matrices as introduced in Section 2. Further, the product $(A_i^j) \cdot (B_k^\ell)$ of matrices A and B (A's column number equals B's row number) is $\Gamma_k^j(A \otimes B)$, where $A \otimes B$ is considered as a $(2,2)$-tensor. The rearrangement is easily seen to be computable in polytime.

From this we can conclude, that a tensor formula can be translated into a tensor expression, in which each tensor product is followed by a junction. Hence, the non-zero tensor formula problem reduces to the non-zero tensor *expression* problem. On the other hand, generalizing the above embedding of a $(2,2)$-tensor into a matrix, arbitrary tensors can be embedded into matrices by listing the entries row- and column-wise following the rule "run through later indices first." Moreover, tensor operations can be translated into corresponding matrix operations in much the same way as is done above the other way around (see [18]) and it is easy to see, that also this translation can be performed in polytime. It follows, that the non-zero tensor expression problem reduces to the non-zero tensor formula problem and, hence, the problems are computationally equivalent.