Noname manuscript No.
(will be inserted by the editor)

Processing Succinct Matrices and Vectors

Markus Lohrey · Manfred Schmidt-Schauß

Received: date / Accepted: date

Abstract We study the complexity of algorithmic problems for matrices that are represented by multi-terminal decision diagrams (MTDD). These are a variant of ordered decision diagrams, where the terminal nodes are labeled with arbitrary elements of a semiring (instead of 0 and 1). A simple example shows that the product of two MTDD-represented matrices cannot be represented by an MTDD of polynomial size. To overcome this deficiency, we extended MTDDs to +MTDDs by allowing componentwise symbolic addition of variables (of the same dimension) in rules. It is shown that accessing an entry, equality checking, matrix multiplication, and other basic matrix operations can be solved in polynomial time for +MTDD-represented matrices. On the other hand, testing whether the determinant of a MTDD-represented matrix vanishes is PSPACE-complete, and the same problem is NP-complete for +MTDD-represented diagonal matrices. Computing a specific entry in a product of MTDD-represented matrices is #P-complete.

1 Introduction

Algorithms that work on succinct representations of certain objects can nowadays be found in many areas of computer science. A paradigmatic example is the use of OBDDs (ordered binary decision diagrams) in hardware verification [7,30]. OBDDs are a succinct representation of Boolean functions. Consider a

The first (second) author is supported by the DFG grant LO 748/8-2 (SCHM 986/9-2).

Markus Lohrey
Universität Siegen, Department für Elektrotechnik und Informatik, D-57076 Siegen, Germany
E-mail: lohrey@eti.uni-siegen.de

Manfred Schmidt-Schauß
Institut für Informatik, Goethe-Universität, D-60054 Frankfurt, Germany
E-mail: Manfred Schmidt-Schauss ¡schauss@ki.informatik.uni-frankfurt.de

boolean function $f(x_1, \dots, x_n)$ in n input variables. One can represent f by its decision tree, which is a full binary tree of height n with $\{0, 1\}$-labelled leaves. The leaf that is reached from the root via the path $(a_1, \dots, a_n) \in \{0, 1\}^n$ (where $a_i = 0$ means that we descend to the left child in the i-th step, and $a_i = 1$ means that we descend to the right child in the i-th step) is labelled with the bit $f(a_1, \dots, a_n)$. This decision tree can be folded into a directed acyclic graph by eliminating repeated occurrences of isomorphic subtrees. The result is the OBDD for f with respect to the variable ordering $x_1, \dots, x_n$.[1] Bryant was the first who realized that OBDDs are an adequate tool in order to handle the state explosion problem in hardware verification [7].

OBDDs can be also used for storing large graphs. A graph G with 2^n nodes and adjacency matrix M_G can be represented by the boolean function $f_G(x_1, y_1, \dots, x_n, y_n)$, where $f_G(a_1, b_1, \dots, a_n, b_n)$ is the entry of M_G at position (a, b); here $a_1 \cdots a_n$ ($b_1 \cdots b_n$, respectively) is the binary representation of the index a (b, respectively). Note that we use the so called interleaved variable ordering here, where the bits of the two coordinates a and b are bitwise interleaved. This ordering turned out to be convenient in the context of OBDD-based graph representation, see e.g. [13].

Classical graph problems (like reachability, alternating reachability, existence of a Hamiltonian cycle) have been studied for OBDD-represented graphs in [12, 40]. It turned out that these problems are exponentially harder for OBDD-represented graphs than for explicitly given graphs. In [40] an upgrading theorem for OBDD-represented graphs was shown. It roughly states that completeness of a problem A for a complexity class C under quantifier free reductions implies completeness of the OBDD-variant of A for the exponentially harder version of C under polynomial time reductions.

In the same way as OBDDs represent boolean mappings, functions from $\{0, 1\}^n$ to any set S can be represented. One simply has to label the leaves of the decision tree with elements from S. This yields multi-terminal decision diagrams (MTDDs) [14]. Of particular interest is the case, where S is a semiring, e.g. $\mathbb{N}$ or $\mathbb{Z}$. In the same way as an adjacency matrix (i.e., a boolean matrix) of dimension 2^n can be represented by an OBDD, a matrix of dimension 2^n over any semiring can be represented by an MTDD. As for OBDDs, we assume that the bits of the two coordinates a and b are interleaved in the order $a_1, b_1, \dots, a_n, b_n$. This implies that an MTDD can be viewed as a set of rules of the form

$$A \to \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \qquad \text{or} \qquad B \to a \quad \text{with} \quad a \in S, \tag{1}$$

where A, $A_{1,1}$, $A_{1,2}$, $A_{2,1}$, and $A_{2,2}$ are variables that correspond to certain nodes of the MTDD (namely those nodes that have even distance from the root node). Every variable produces a matrix of dimension 2^h for some $h \geq 0$, and this value h is called the height of the variable. The variables $A_{i,j}$ in (1)

[1] Here, we are cheating a bit: In OBDDs a second elimination rule is applied that removes nodes for which the left and right child are identical. On the other hand, it is known that asymptotically the compression achieved by this elimination rule is negligible [41].

must have the same height h, and A has height $h+1$. The variable B has height 0. We assume that the additive monoid of the semiring S is finitely generated. Hence every $a \in S$ has a finite representation as a word over the generators.

MTDDs yield very compact representations of sparse matrices. It was shown that an $(n \times n)$-matrix with m nonzero entries can be represented by an MTDD of size $O(m \log n)$ [14, Theorem 3.2], which is better than standard succinct representations for sparse matrices. Moreover, MTDDs can also yield very compact representations of non-sparse matrices. For instance, the Hadamard matrix of dimension 2^n can be represented by an MTDD of size $O(n)$, see [14] and Example 3. In fact, the usual recursive definition of the n-th Hadamard matrix is exactly an MTDD.

Matrix algorithms for MTDDs are studied in [14] as well, but no precise complexity analysis is carried out. In fact, the straightforward matrix multiplication algorithm for multi-terminal decision diagrams from [14] has an exponential worst case running time, and this is unavoidable: The smallest MTDD that produces the product of two MTDD-represented matrices may be of exponential size in the two MTDDs, see Theorem 6. The first main contribution of this paper is a generalization of MTDDs that overcomes this deficiency: A +MTDD consists of rules of the form (1) together with addition rules of the form $A \to B + C$, where "+" refers to matrix addition over the underlying semiring. There, A, B, and C must have the same height, i.e., produce matrices of the same dimension. We show that a +MTDD for the product of two +MTDD-represented matrices can be computed in polynomial time (Theorem 7). In Section 4.1 we also present efficient (polynomial time) algorithms for several other important matrix problems on +MTDD-represented input matrices: computation of a specific matrix entry, computation of the trace, matrix transposition, tensor and Hadamard product. Section 5 deals with equality checking. It turns out that equality of +MTDD-represented matrices can be checked in polynomial time, if the additive monoid is cancellative, in all other cases equality checking is coNP-complete.

To the knowledge of the authors, complexity results similar to those from [12,40] for OBDDs do not exist in the literature on MTDDs. Our second main contribution fills this gap. We prove that already for MTDDs over $\mathbb{Z}$ it is PSPACE-complete to check whether the determinant of the generated matrix is zero (Theorem 18). This result is shown by lifting a classical construction of Toda [36] (showing that computing the determinant of an explicitly given integer matrix is complete for the counting class GapL) to configuration graphs of polynomial space bounded Turing machines, which are of exponential size. It turns out that the adjacency matrix of the configuration graph of a polynomial space bounded Turing machine can be produced by a small MTDD. Theorem 18 sharpens a recent result from [18] stating that it is PSPACE-complete to check whether the determinant of a matrix that is represented by a boolean circuit (see Section 4.2) vanishes. We also prove several hardness results for counting classes. For instance, computing a specific entry of a matrix power A^n, where A is given by an MTDD over $\mathbb{N}$ is #P-complete (#PSPACE-complete, respectively) if n is given unary (binary, respectively). Here, #P (#PSPACE, respectively) is the class of functions counting the number of accepting computations of a nondeterministic polynomial time Turing machine [39] (a nondeterministic polynomial space Turing machine [22], respectively).

2 Related work

Sparse matrices and quad-trees. To the knowledge of the authors, most of the literature on matrix compression deals with sparse matrices, where most of the matrix entries are zero. There are several succinct representations of sparse matrices. One of them are *quad-trees*, which are used in computer graphics for the representation of large constant areas in 2-dimensional pictures, see for example [33,11]. Actually, an MTDD can be seen as a quad-tree that is folded into a dag by merging identical subtrees.

Two-dimensional straight-line programs. MTDDs are also a special case of 2-dimensional straight-line programs (SLPs). A (1-dimensional) SLP is a context-free grammar in Chomsky normal form that generates exactly one string. An SLP with n rules can generate a string of length 2^n. Therefore an SLP can be seen as a succinct representation of the string it generates. Algorithmic problems that can be solved efficiently (in polynomial time) on SLP-represented strings are for instance equality checking (first shown by Plandowski [32]) and pattern matching, see [26] for a survey.

In [4] a 2-dimensional extension of SLPs (2SLPs in the following) was defined. Here, every variable of the grammar generates a (not necessarily square) matrix (or picture), where every position is labeled with an alphabet symbol. Moreover, there are two (partial) concatenation operations: horizontal composition (which is defined for two pictures if they have the same height) and vertical composition (which is defined for two pictures if they have the same width). An essentially equivalent formalism was introduced in [19] under the term context-free rectangular image grammar. Unfortunately, 2SLPs do not share all the nice algorithmic properties of (1-dimensional) SLPs: Berman et al. [4] proved that testing whether two 2SLPs produce the same picture belongs to coRP (co-randomized polynomial time), and recently the upper bound was improved to coRNC^2 [21], but no deterministic polynomial time algorithm is known for this problem. Moreover, checking whether an explicitly given (2SLP-represented, respectively) picture appears within a 2SLP-represented picture is NP-complete (Σ_2^P-complete, respectively) [4]. Related hardness results in this direction concern the convolution of two SLP-represented strings of the same length (which can be seen as a picture of height 2). The convolution of strings $u = a_1 \cdots a_n$ and $v = b_1 \cdots b_n$ is the string $(a_1, b_1) \cdots (a_n, b_n)$. By a result from [5] (which is stated in terms of the related operation of literal shuffle), the size of a shortest SLP for the convolution of two strings that are given by SLPs G and H can be exponential in the size of G and H. Moreover, it is PSPACE-complete to check for two SLP-represented strings u and v and an NFA T operating on strings of pairs of symbols, whether T accepts the convolution of u and v [25].

MTDDs restrict 2SLPs by forbidding unbalanced derivation trees. The derivation tree of an MTDD results from unfolding the rules in (1); it is a tree, where every non-leaf node has exactly four children and every root-leaf path has the same length.

Let us finally mention that straight-line programs are also used for the compact representation of other objects, e.g. polynomials [20], trees [27], graphs [23], and regular languages [17].

Tensor circuits. In [3, 10], the problems of evaluating tensor formulas and tensor circuits were studied. Let us restrict to tensor circuits. A tensor circuit is a circuit where the gates evaluate to matrices over a semiring and the following operations are used: matrix addition, matrix multiplication, and tensor product. Recall that the tensor product of two matrices $A = (a_{i,j})_{1 \le i \le m, 1 \le i \le m}$ and B is the matrix

$$A \otimes B = \begin{pmatrix} a_{1,1}B & \cdots & a_{1,m}B \\ \vdots & & \vdots \\ a_{n,1}B & \cdots & a_{n,m}B \end{pmatrix}$$

It is a $(mk \times nl)$-matrix if B is a $(k \times l)$-matrix. In [3] it is shown among other results that computing the output value of a scalar tensor circuit (i.e., a tensor circuit that yields a (1×1)-matrix) over the natural numbers is complete for the counting class #EXP, and the restriction to so called tame circuits is #PSPACE-complete. A +MTDD over $\mathbb{Z}$ can be seen as a tensor circuit that (i) does not use matrix multiplication and (ii) where for every tensor product the left factor is a (2×2)-matrix. To see the correspondence, note that

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes A_{1,1} + \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \otimes A_{1,2} + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \otimes A_{2,1} + \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes A_{2,2}$$

$$\begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \otimes B = \begin{pmatrix} a_{1,1}B & a_{1,2}B \\ a_{2,1}B & a_{2,2}B \end{pmatrix}$$

Each of the matrices $a_{i,j}B$ can be generated from B and $-B$ using $\log |a_{i,j}|$ many additions (here we use the fact that the underlying semiring is $\mathbb{Z}$).

Counting classes. There is a large number of #P-complete counting problems. On the other hand, only a few number of papers deals with the class #PSPACE and counting problems that are complete for that class. An example of a natural #PSPACE-complete counting problem is counting the number of strings not accepted by a given NFA [22]. Recently, it was shown that counting the number of word models of a given formula of linear time temporal logic, where the length of the word models is specified in unary (binary, respectively) notation is complete for #P (#PSPACE, respectively) [38].

3 Preliminaries

We consider matrices over a semiring $(S, +, \cdot)$ with $(S, +)$ a finitely generated commutative monoid with unit 0. The unit of the monoid $(S, \cdot)$ is 1. We assume that $0 \cdot a = a \cdot 0 = 0$ for all $a \in S$. Hence, if $|S| > 1$, then $1 \neq 0$ ($0 = 1$ implies $a = 1 \cdot a = 0 \cdot a = 0$ for all $a \in S$). With $S^{n \times n}$ we denote the set of all $(n \times n)$-matrices over S.

All time bounds in this paper implicitly refer to the RAM model of computation with a logarithmic cost measure for arithmetical operations on integers, where arithmetic operations on n-bit numbers need time $O(n)$. For a number $n \in \mathbb{Z}$ let us denote with $\mathsf{bin}(n)$ its binary encoding.

We assume that the reader has some basic background in complexity theory, in particular we assume that the reader is familiar with the classes NP, coNP, and PSPACE. With polyL (polylogarithmic space) we denote the class $\bigcup_{k \geq 1} \mathsf{DSPACE}(\log^k(n))$, which by Savitch's theorem is equal to the class $\bigcup_{k \geq 1} \mathsf{NSPACE}(\log^k(n))$. A function $f : \{0,1\}^* \to \{0,1\}^*$ belongs to the class $\mathsf{FSPACE}(s(n))$ ($\mathsf{FTIME}(s(n))$, respectively) if f can be computed on a deterministic Turing machine in space (time, respectively) $s(n)$.[2] As usual, only the space on the working tapes is counted. Moreover, the output is written from left to right on the output tape, i.e., in each step the machine either outputs a new symbol on the output tape, in which case the output head moves one cell to the right, or the machine does not output a new symbol in which case the output head does not move. We define

$$\begin{aligned} \mathsf{FP} &= \bigcup_{k \geq 1} \mathsf{FTIME}(n^k), \\ \mathsf{FpolyL} &= \bigcup_{k \geq 1} \mathsf{FSPACE}(\log^k(n)), \\ \mathsf{FPSPACE} &= \bigcup_{k \geq 1} \mathsf{FSPACE}(n^k). \end{aligned}$$

Note that for a function $f \in \mathsf{FPSPACE}$ we have $|f(w)| \leq 2^{|w|^{O(1)}}$ for every input. The function that maps an explicitly given integer matrix (with binary encoded entries) to its determinant belongs to uniform NC^2 [9] and hence to $\mathsf{FSPACE}(\log^2(n))$.

We need the following simple lemma, see e.g. [28, Lemma 1].

Lemma 1 *If $f \in \mathsf{FPSPACE}$ and $L \in \mathsf{polyL}$ then $f^{-1}(L) \in \mathsf{PSPACE}$.*

The following result can be shown in the same way as Lemma 1:

Lemma 2 *If $f \in \mathsf{FPSPACE}$ and $g \in \mathsf{FpolyL}$ then the mapping h defined by $h(x) = g(f(x))$ for all inputs x belongs to $\mathsf{FPSPACE}$.*

[2] The assumption that the input and output alphabet of f is binary is made here to make the definitions more readable; the extension to arbitrary finite alphabets is straightforward.

The counting class $\#\mathsf{P}$ consists of all functions $f : \{0,1\}^* \to \mathbb{N}$ for which there exists a nondeterministic polynomial time Turing machine M with input alphabet $\{0,1\}$ such that for all $x \in \{0,1\}^*$, $f(x)$ is the number of accepting computation paths of M for input x. If we replace nondeterministic polynomial time Turing machines by nondeterministic polynomial space Turing machines (nondeterministic logspace Turing machines, respectively), we obtain the class $\#\mathsf{PSPACE}$ [22] ($\#\mathsf{L}$ [2], respectively). Note that for a mapping $f \in \#\mathsf{PSPACE}$, the number $f(x)$ may grow doubly exponential in $|x|$, whereas for $f \in \#\mathsf{P}$, the number $f(x)$ is bounded singly exponential in $|x|$. Ladner [22] has shown that a mapping $f : \{0,1\}^* \to \mathbb{N}$ belongs to $\#\mathsf{PSPACE}$ if and only if the mapping $x \mapsto \mathsf{bin}(f(x))$ belongs to $\mathsf{FPSPACE}$. One cannot expect a corresponding result for the class $\#\mathsf{P}$: If for every function $f \in \#\mathsf{P}$ the mapping $x \mapsto \mathsf{bin}(f(x))$ belongs to FP, then by Toda's theorem [37] the polynomial time hierarchy collapses down to P. For $f \in \#\mathsf{L}$, the mapping $x \mapsto \mathsf{bin}(f(x))$ belongs to NC^2 and hence to $\mathsf{FP} \cap \mathsf{FSPACE}(\log^2(n))$ [2, Theorem 4.1]. The class GapL (GapP, $\mathsf{GapPSPACE}$, respectively) consists of all differences of two functions in $\#\mathsf{L}$ ($\#\mathsf{P}$, $\#\mathsf{PSPACE}$, respectively). From Ladner's result [22] it follows easily that a function $f : \{0,1\}^* \to \mathbb{Z}$ belongs to $\mathsf{GapPSPACE}$ if and only if the mapping $x \mapsto \mathsf{bin}(f(x))$ belongs to $\mathsf{FPSPACE}$, see also [15, Theorem 6].

Logspace reductions between functions can be defined analogously to the language case: If $f, g : \{0,1\}^* \to X$ with $X \in \{\mathbb{N}, \mathbb{Z}\}$, then f is logspace reducible to g if there exists a function $h \in \mathsf{FSPACE}(\log n)$ such that $f(x) = g(h(x))$ for all x. Toda [36] has shown that computing the determinant of a given integer matrix is GapL-complete.

4 Succinct matrix representations

In this section, we introduce several succinct matrix representations. We formally define multi-terminal decision diagrams and their extension by the addition operation. Moreover, we briefly discuss the representation of matrices by boolean circuits.

4.1 Multi-terminal decision diagrams

Fix a semiring $(S, +, \cdot)$ with $(S, +)$ a finitely generated commutative monoid, and let $\Gamma \subseteq S$ be a finite generating set for $(S, +)$. Thus, every element of S can be written as a finite sum $\sum_{a \in \Gamma} n_a a$ with $n_a \in \mathbb{N}$. A *multi-terminal decision diagram G with addition* (+MTDD) *of height h* is a triple (N, P, A_0), where N is a finite set of variables which is partitioned into non-empty sets N_i $(0 \le i \le h)$, $N_h = \{A_0\}$ (A_0 is called the *start variable*), and P is a set of rules, each having one of the following three forms:

- $A \to \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}$ with $A \in N_i$ and $A_{1,1}, A_{1,2}, A_{2,1}, A_{2,2} \in N_{i-1}$ for some $1 \le i \le h$

- $A \to A_1 + A_2$ with $A, A_1, A_2 \in N_i$ for some $0 \leq i \leq h$
- $A \to a$ with $A \in N_0$ and $a \in \Gamma \cup \{0\}$

Moreover, for every variable $A \in N$ there is exactly one rule with left-hand side A, and the relation $\{(A, B) \in N \times N \mid B$ occurs in the right-hand side for $A\}$ is acyclic. If $A \in N_i$ then we say that A has height i. The +MTDD G is called an MTDD if for every addition rule $(A \to A_1 + A_2) \in P$ we have $A, A_1, A_2 \in N_0$. In other words, only scalars are allowed to be added. Since we assume that $(S, +)$ is generated by Γ, this allows to produce arbitrary elements of S as matrix entries. For every $A \in N_i$ we define a square matrix $\mathsf{val}(A)$ of dimension 2^i in the obvious way by unfolding the rules:

$$\mathsf{val}(A) = \begin{pmatrix} \mathsf{val}(A_{1,1}) & \mathsf{val}(A_{1,2}) \\ \mathsf{val}(A_{2,1}) & \mathsf{val}(A_{2,2}) \end{pmatrix} \text{ if } \left(A \to \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}\right) \in P$$
$$\mathsf{val}(A) = \mathsf{val}(A_1) + \mathsf{val}(A_2) \text{ if } (A \to A_1 + A_2) \in P$$
$$\mathsf{val}(A) = a \text{ if } (A \to a) \in P$$

Moreover, let $\mathsf{val}(G) = \mathsf{val}(A_0)$ for the start variable A_0 of G. This is a $(2^h \times 2^h)$-matrix. The size of a rule $A \to a$ with $a \in \Gamma \cup \{0\}$ is 1, all other rules have size $\log |N|$. The size $|G|$ of the +MTDD G is the sum of the sizes of its rules; this is up to constant factors the length of the binary coding of G. A +MTDD G of size $O(n \log n)$ can represent a $(2^n \times 2^n)$-matrix. Note that only square matrices whose dimension is a power of 2 can be represented. Matrices not fitting this format can be filled up appropriately (for example with zeroes, or with ones on the diagonal), depending on the purpose.

In the same way as 2-dimensional matrices, one can also produce 1-dimensional vectors with +MTDDs. In this case, we use rules of the form $A \to (A_1, A_2)$ to double the dimension. We will refer to 1-dimensional +MTDDs in this case. In fact, the notion of a +MTDD makes sense in any dimension.

An MTDD, where all rules have the form $A \to a \in \Gamma \cup \{0\}$ or $A \to B + C$ generates an element of the semiring S. Such an MTDD is an arithmetic circuit in which only input gates and addition gates are used, and is called a *+-circuit* in the following. In case the underlying semiring is $\mathbb{Z}$, a +-circuit with n variables can produce a number of size 2^n, and the binary encoding of this number can be computed in time $\mathcal{O}(n^2)$ from the +-circuit (since, we need n additions of numbers with at most n bits). In general, for a +-circuit over the semiring S, we can compute in quadratic time numbers n_a $(a \in \Gamma)$ such that $\sum_{a \in \Gamma} n_a \cdot a$ is the semiring element to which the +-circuit evaluates to.

Note that the notion of a +MTDD makes sense for commutative monoids, since we only used the addition of the underlying semiring. But soon, we want to multiply matrices, for which we need a semiring.

Example 3 The n-th Hadamard matrix is produced by the following MTDD, where H_n is the start variable:

$$\begin{aligned} &H_0 \to 1, && H'_0 \to -1, \\ &H_i \to \begin{pmatrix} H_{i-1} & H_{i-1} \\ H_{i-1} & H'_{i-1} \end{pmatrix}, && H'_i \to \begin{pmatrix} H'_{i-1} & H'_{i-1} \\ H'_{i-1} & H_{i-1} \end{pmatrix} \quad (1 \le i \le n) \end{aligned}$$

Note that H'_i produces $-\mathsf{val}(H_i)$.

Example 4 It is straightforward to produce the unit matrix I_{2^n} of dimension 2^n by an MTDD of size $O(n \log n)$:

$$A_0 \to 1, \quad 0_0 \to 0, \quad A_j \to \begin{pmatrix} A_{j-1} & 0_{j-1} \\ 0_{j-1} & A_{j-1} \end{pmatrix}, \quad 0_j \to \begin{pmatrix} 0_{j-1} & 0_{j-1} \\ 0_{j-1} & 0_{j-1} \end{pmatrix} \quad (1 \le j \le n).$$

The start variable is A_n here.

The lower triangular $(2^n \times 2^n)$-matrix, where entries on the diagonal and below are 1 is generated by the above rules for the variables 0_j together with the rules

$$T_0 \to 1, \quad 1_0 \to 1, \quad T_j \to \begin{pmatrix} T_{j-1} & 0_{j-1} \\ 1_{j-1} & T_{j-1} \end{pmatrix}, \quad 1_j \to \begin{pmatrix} 1_{j-1} & 1_{j-1} \\ 1_{j-1} & 1_{j-1} \end{pmatrix} \quad (1 \le j \le n).$$

To produce the $(2^n \times 2^n)$-matrix over $\mathbb{Z}$, where all entries in the k-th row are k, we need the following rules:

$$\begin{aligned} &E_0 \to 1, \quad E_j \to \begin{pmatrix} E_{j-1} + E_{j-1} & E_{j-1} + E_{j-1} \\ E_{j-1} + E_{j-1} & E_{j-1} + E_{j-1} \end{pmatrix} \quad (1 \le j \le n) \\ &C_0 \to 1, \quad C_j \to \begin{pmatrix} C_{j-1} & C_{j-1} \\ C_{j-1} + E_{j-1} & C_{j-1} + E_{j-1} \end{pmatrix} \quad (1 \le j \le n). \end{aligned}$$

Here, we are a bit more liberal with respect to the format of rules, but the above rules can be easily brought into the form from the general definition of a +MTDD. Note that E_j generates the $(2^j \times 2^j)$-matrix with all entries equal to 2^j, and that C_n generates the desired matrix.

Note that the matrix from the last example cannot be produced by an MTDD of polynomial size, since it contains an exponential number of different matrix entries (for the same reason it cannot be produced by a 2SLP [4]). This holds for any non-trivial semiring.

Theorem 5 *For any semiring S with at least two elements, +MTDDs are exponentially more succinct than MTDDs.*

Proof For simplicity we argue with 1-dimensional MTDDs, which generate vectors. We must have $1 \neq 0$ in S. Let $m, d > 0$ be such that $m = 2^d$. We define a +MTDD as follows:

For $0 \le i \le m-1$ let T_i be a variable such that $\mathsf{val}(T_i)$ has length m, the i-th entry is 1 (the first entry is the 0-th entry) and all other entries are 0.

Moreover, for $0 \leq i \leq m-1$ let the variable U_i be such that $\mathsf{val}(U_i)$ is the concatenation of 2^i copies of $\mathsf{val}(T_i)$. Let V_0 produce the 0-vector of length $m = 2^d$, and for $0 \leq i \leq m-1$ let $V_{i+1} \to (V_i, V_i + U_i)$. Then $\mathsf{val}(V_m)$ is of length 2^{d+m} and consists of the concatenation of all binary strings of length m. The +MTDD for this vector contains $O(m^2)$ many variables and has size $O(m^2 \log m)$. On the other hand, an equivalent MTDD must have size at least 2^m, since for every binary string of length m there must exist a variable. □

The following result shows that the matrix product of two MTDD-represented matrices may be incompressible with MTDDs.

Theorem 6 *For any semiring S with at least two elements there exist MTDDs G_n and H_n of the same height n and size $O(n^2 \log n)$ such that $\mathsf{val}(G_n) \cdot \mathsf{val}(H_n)$ can only be represented by an MTDD of size at least 2^n.*

Proof The construction is similar to those in the proof of Theorem 5. We must have $0 \neq 1$ in S. Let $m = 2^d$. For $0 \leq i \leq m-1$ let A_i be such that $\mathsf{val}(A_i)$ is the $(m \times m)$-matrix with $\mathsf{val}(A_i)_{1,i+1} = 1$ and all other entries 0. Define $B_{i,0}$ by $B_{i,0} \to A_i$ and

$$B_{i,j} \to \begin{pmatrix} B_{i,j-1} & B_{i,j-1} \\ 0 & 0 \end{pmatrix}$$

for $1 \leq j \leq i$. Then $\mathsf{val}(B_{i,i})$ is the $(2^{d+i} \times 2^{d+i})$-matrix, where the first row is the vector $\mathsf{val}(U_i)$ from the proof of Theorem 5, and all other entries are 0. Finally add variables $C_0, \ldots, C_m$, where $\mathsf{val}(C_0)$ is the $(m \times m)$-matrix with all entries 0 and

$$C_{i+1} \to \begin{pmatrix} C_i & C_i \\ 0 & B_{i,i} \end{pmatrix}$$

$0 \leq i \leq m-1$. In this way we obtain an MTDD of size $O(m^2 \log m)$ for the $(2^{m+d} \times 2^{m+d})$-matrix $\mathsf{val}(C_m)$. This matrix has the following properties:

- Only the entries 0 and 1 appear in $\mathsf{val}(C_m)$.
- Every column of $\mathsf{val}(C_m)$ contains at most one entry that is 1.
- The i-th column of $\mathsf{val}(C_m)$ contains an entry that is 1 if and only if the i-th entry in the vector $\mathsf{val}(V_m)$ from the proof of Theorem 5 is 1.

Hence, the product of the $(2^{m+d} \times 2^{m+d})$-matrix, where every entry is 1, with the matrix $\mathsf{val}(C_m)$ is the matrix, where every row is the vector $\mathsf{val}(V_m)$ from the proof of Theorem 5. □

On the other hand, the product of two +MTDD-represented matrices can be represented by a polynomially sized +MTDD:

Theorem 7 *For +MTDDs G_1 and G_2 of the same height one can compute in time $O(|G_1| \cdot |G_2|)$ a +MTDD G of size $O(|G_1| \cdot |G_2|)$ such that $\mathsf{val}(G) = \mathsf{val}(G_1) \cdot \mathsf{val}(G_2)$.*

Proof Recall that Γ is a finite generating set for the additive monoid of our underlying semiring S. For all pairs $(a,b) \in (\Gamma \cup \{0\}) \times (\Gamma \cup \{0\})$, we can write down a $+$-circuit of constant size that computes ab. Let $S_{a,b}$ denote its start variable.

Given two +MTDDs G_1 and G_2, we compute a new +MTDD G that contains for all variables A of G_1 and B of G_2 of the same height a variable (A,B) such that $\mathsf{val}_G(A,B) = \mathsf{val}_{G_1}(A)\cdot\mathsf{val}_{G_2}(B)$. So, let A and B be variables of G_1 and G_2, respectively, of the same height.

1. If A and B are of height 0 and the corresponding rules are $A \to a$, $B \to b$ with $a,b \in \Gamma \cup \{0\}$, then the rule for (A,B) is $(A,B) \to S_{a,b}$ (actually, we should replace $S_{a,b}$ by its corresponding right-hand side).
2. If the rule for A is of the form $A \to A_1 + A_2$, then we add the rule $(A,B) \to (A_1,B) + (A_2,B)$ to G.
3. If the right-hand side for A is not a sum but the rule for B is of the form $B \to B_1 + B_2$, then we add the rule $(A,B) \to (A,B_1) + (A,B_2)$ to G.
4. Finally, assume that neither the right-hand side for A nor for B is a sum or from $\Gamma \cup \{0\}$. Then the rules for A and B have the form

$$A \to \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \text{ and } B \to \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix}.$$

Then we add the following rules to G:

$$C_{i,j} \to (A_{i,1}, B_{1,j}) + (A_{i,2}, B_{2,j}) \text{ for } 1 \le i,j \le 2$$
$$(A,B) \to \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

Clearly, if S_i is the start variable of G_i, then $\mathsf{val}_G(S_1,S_2) = \mathsf{val}(G_1)\cdot\mathsf{val}(G_2)$. The bounds from the theorem for the construction and size of G follow immediately from the construction. □

The following proposition presents several further matrix operations that can be easily implemented in polynomial time for +MTDD-represented input matrices.

Proposition 8 *Let G and H be +MTDDs of heights g and h, respectively, with $|G| = n$, $|H| = m$, and $1 \le i,j \le 2^g$.*

(1) A +MTDD for the transposition of $\mathsf{val}(G)$ can be computed in time $O(n)$.
(2) +-circuits for the sum of all entries of $\mathsf{val}(G)$ and the trace of $\mathsf{val}(G)$ can be computed in time $O(n)$.
(3) A +-circuit for the matrix entry $\mathsf{val}(G)_{i,j}$ can be computed in time $O(n)$.
(4) +MTDDs of size $O(n \cdot m)$ for the tensor product $\mathsf{val}(G) \otimes \mathsf{val}(H)$ (which includes the scalar product) and the element-wise (Hadamard) product $\mathsf{val}(G) \circ \mathsf{val}(H)$ (assuming $g = h$) can be computed in time $O(n \cdot m)$.

Proof Point (1) (transposition): We replace every rule in G of the form

$$A \to \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \tag{2}$$

by the rule

$$A \to \begin{pmatrix} A_{1,1} & A_{2,1} \\ A_{1,2} & A_{2,2} \end{pmatrix}.$$

Point (2): The sum of all entries of $\mathsf{val}(G)$ can be represented by the $+$-circuit that contains all rules $A \to A_{1,1} + A_{1,2} + A_{2,1} + A_{2,2}$ for G-rules of the form (2). Similarly, we can compute a $+$-circuit for the trace of $\mathsf{val}(G)$ by replacing every rule (2) by $A \to A_{1,1} + A_{2,2}$.

Point (3): We transform the $+$MTDD G into a $+$-circuit G' with the same set of variables such that $\mathsf{val}(G') = (\mathsf{val}(G))_{i,j}$. Let $(i_g \cdots i_1)$ and $(j_g \cdots j_1)$ be the binary expansions of $i-1$ and $j-1$ (numbers in the range $[0, 2^g - 1]$), respectively, where i_g and j_g are the most significant bits. Here, we add leading zeros on the left so that both numbers have exactly g bits.

Now we can define the rules of the $+$-circuit G'. Rules of the form $A \to a$ with a a semigroup generator and $A \to A_1 + A_2$ are simply copied to G'. For a rule of the form

$$A \to \begin{pmatrix} A_{0,0} & A_{0,1} \\ A_{1,0} & A_{1,1} \end{pmatrix},$$

where A has height k we add to G' the rule $A \to A_{i_k, j_k}$.

Point (4): For every variable C of G and every variable D of H let (C, D) be a new variable of height $\mathsf{height}(C) + \mathsf{height}(D)$. We define the rule for (C, D) in such a way that $\mathsf{val}(C, D) = \mathsf{val}(C) \otimes \mathsf{val}(D)$. The rules reflect the bilinearity of the tensor product.

If $C \to a$ and $D \to b$ for $a, b \in \Gamma$, then $(C, D) \to S_{a,b}$, where $S_{a,b}$ is the start variable for a (constant size) $+$-circuit that computes $a \cdot b$.

Now assume that $C \to a$ but the rule for D is not terminal. If $D \to D_1 + D_2$, then $(C, D) \to (C, D_1) + (C, D_2)$ and if

$$D \to \begin{pmatrix} D_{1,1} & D_{1,2} \\ D_{2,1} & D_{2,2} \end{pmatrix}$$

then

$$(C, D) \to \begin{pmatrix} (C, D_{1,1}) & (C, D_{1,2}) \\ (C, D_{2,1}) & (C, D_{2,2}) \end{pmatrix}.$$

Finally, assume that the rule for C is not terminal. If $C \to C_1 + C_2$, then $(C, D) \to (C_1, D) + (C_2, D)$, and if

$$C \to \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix},$$

then

$$(C, D) \to \begin{pmatrix} (C_{1,1}, D) & (C_{1,2}, D) \\ (C_{2,1}, D) & (C_{2,2}, D) \end{pmatrix}.$$

The proof for the construction of the element-wise product is similar as for the tensor-product. □

4.2 Boolean circuits

In [16] boolean circuits were used for the succinct representation of strings and graphs. A boolean circuit with n inputs represents a binary string of length 2^n, namely the string of output values for the 2^n many input assignments (concatenated in lexicographic order). In a similar way, we can use circuits to encode large matrices. Here, we restrict to matrices over $\mathbb{Z}$. We propose two alternatives:

Definition 9 A boolean circuit $C(\overline{x}, \overline{y}, \overline{z})$ with $|\overline{x}| = m$ and $|\overline{y}| = |\overline{z}| = n$ encodes a $(2^n \times 2^n)$-matrix $M_{C,2}$ with entries of absolute value at most 2^{2^m-1} that is defined as follows: For all $\overline{a} \in \{0,1\}^m \setminus \{(0,\dots,0)\}$ and $\overline{b}, \overline{c} \in \{0,1\}^n$, the $\overline{a}$-th bit (in lexicographic order) of the matrix entry at position $(\overline{b}, \overline{c})$ in $M_{C,2}$ is 1 if and only if $C(\overline{a}, \overline{b}, \overline{c}) = 1$. Moreover, the sign of this entry is $+1$ if $C(0,\dots,0,\overline{b},\overline{c}) = 1$, otherwise the sign is -1.

Note that in contrast to +MTDDs, the size of an entry in $M_{C,2}$ can be doubly exponential in the size of the representation C (this is the reason for the index 2 in $M_{C,2}$). The following alternative, which is also used in [29], is closer to +MTDDs:

Definition 10 A boolean circuit $C(\overline{x}, \overline{y})$ with $|\overline{x}| = |\overline{y}| = n$ and m output gates encodes a $(2^n \times 2^n)$-matrix $M_{C,1}$ with entries of absolute value at most 2^{m-1} that is defined as follows: For all $\overline{a}, \overline{b} \in \{0,1\}^n$, $C(\overline{a}, \overline{b})$ is the binary encoding of the entry at position $(\overline{a}, \overline{b})$ in $M_{C,1}$, where the sign is encoded by the first output bit.

Circuit representations for matrices are at least as succinct as +MTDDs. More precisely, from a given +MTDD G one can compute in logspace a Boolean circuit C such that $M_{C,1} = \mathsf{val}(G)$. This is a direct corollary of Proposition 8(3) (stating that a given entry of an +MTDD-represented matrix can be computed in polynomial time) and the fact that polynomial time computations can be simulated by boolean circuits.

Recently, it was shown that checking whether for a given circuit C the determinant of the matrix $M_{C,1}$ vanishes is PSPACE-complete [18]. An algebraic version of this result for the algebraic complexity class VPSPACE is shown in [29]. Theorem 18 from Section 6 will strengthen the result from [18] to MTDD-represented matrices.

5 Testing equality

In this section, we consider the problem of testing equality of +MTDD-represented matrices. For this, we do not need the full semiring structure, but we only need the finitely generated additive monoid $(S, +)$. We will show that equality can be checked in polynomial time if $(S, +)$ is cancellative and coNP-complete otherwise.

First we consider the case of a finitely generated abelian group. The proof of the following lemma involves only basic linear algebra.

Lemma 11 *Let $a_{i,1}x_1 + \cdots + a_{i,n}x_n = 0$ $(1 \le i \le n+1)$ be a system of $n+1$ equations over a torsion-free abelian group A, where $a_{i,1}, \ldots, a_{i,n} \in \mathbb{Z}$, and the n variables $x_1, \ldots, x_n$ range over A. One can determine in time polynomial in n and $\max\{\log|a_{i,j}| \mid 1 \le i \le n+1, 1 \le j \le n\}$ an equivalent system of at most n linear equations (having the same solutions in A^n).*

Proof Let $a_i = (a_{i,1}, \ldots, a_{i,n}) \in \mathbb{Z}^n$ be the vector of coefficients of the i-th equation. For $0 \le i \le n$ let $U_i \subseteq \mathbb{Q}^n$ be the subspace of the vector space $\mathbb{Q}^n$ generated by $a_1, \ldots, a_i$ (U_0 is the 0-space). For $i = 1, \ldots, n+1$, we now test whether $a_i \in U_{i-1}$. This can be checked by testing whether a system of linear equations has a solution in $\mathbb{Q}^n$. This problem can be solved in time polynomial in n and $\log(\max\{|a_{k,j}| \mid 1 \le k \le i, 1 \le j \le n\})$, e.g. by Gaussian elimination. If $a_i \in U_{i-1}$ then we obtain an equation

$$\lambda_i a_i = \lambda_1 a_1 + \cdots + \lambda_{i-1} a_{i-1}$$

with $\lambda_1, \ldots, \lambda_i \in \mathbb{Z}$ and $\lambda_i \neq 0$. Hence, if group elements $x_1, \ldots, x_n \in A$ satisfy $a_{j,1}x_1 + \cdots + a_{j,n}x_n = 0$ for all $1 \le j \le i-1$, then we get

$$\lambda_i(a_{i,1}x_1 + \cdots + a_{i,n}x_n) = 0$$

in A. Since A is assumed to be torsion-free, we get $a_{i,1}x_1 + \cdots + a_{i,n}x_n = 0$. Hence, the i-th equation is redundant. Moreover, there must be an $1 \le i \le n+1$ with $a_i \in U_{i-1}$: If $a_i \notin U_{i-1}$ for $1 \le i \le n$, then $a_1, \ldots, a_n$ are linearly independent and therefore generate the full $\mathbb{Q}^n$. But then $a_{n+1} \in U_n$. □

We now prove an analogue of Lemma 11 for abelian groups with torsion. For this, we make use of an efficient algorithm from [34] for computing the so called Howell normal form of a matrix. The precise definition of the Howell normal form is not needed. Recall that the *exponent* of an abelian group A is the smallest integer k (if it exists) such that $kg = 0$ for all $g \in A$.

Lemma 12 *Let $k \ge 2$ and let A be an abelian group of exponent k. Let*

$$a_{i,1}x_1 + \cdots + a_{i,n}x_n = 0 \quad (1 \le i \le n+1)$$

be a system of $n+1$ equations, where $a_{i,1}, \ldots, a_{i,n} \in \mathbb{Z}$, and the n variables $x_1, \ldots, x_n$ range over A. Then one can determine in time polynomial in n, $\log(k)$, and $\max\{\log|a_{i,j}| \mid 1 \le i \le n+1, 1 \le j \le n\}$ an equivalent system of at most n linear equations.

Proof We can consider the coefficients $a_{i,j}$ as elements from $\mathbb{Z}_k$. By [34] we can compute the Howell normal form of the matrix $(a_{i,j})_{1 \le i \le n+1, 1 \le j \le n} \in \mathbb{Z}_k^{(n+1)\times n}$ in polynomial time. The Howell normal form is an $(n \times n)$-matrix with the same row span (a subset of the module $\mathbb{Z}_k^n$) as the original matrix, and hence defines an equivalent set of linear equations. □

Theorem 13 *Let G be a +MTDD over a finitely generated abelian group S. Given two different variables A_1, A_2 of the same height, it is possible to check $\mathsf{val}(A_1) = \mathsf{val}(A_2)$ in time polynomial in $|G|$.*

Proof Since every finitely generated group is a finite direct product of copies of $\mathbb{Z}$ and $\mathbb{Z}_k$ ($k \geq 2$), it suffices to prove the theorem only for these groups.

Consider the case $S = \mathbb{Z}$. The algorithm stores a system of m equations (m will be bounded later) of the form $a_{i,1}B_1 + \cdots + a_{i,k}B_k = 0$, where all $B_1, \ldots, B_k$ are pairwise different variables of the same height h. We treat the variables $B_1, \ldots, B_k$ as variables that range over the torsion-free abelian group $\mathbb{Z}^{2^h \times 2^h}$. We start with the single equation $A_1 - A_2 = 0$. We use the rules of G to transform the system of equations into another system of equations whose variables have strictly smaller height. Assume the current height is $h > 1$. We iterate the following steps until only variables of height $h - 1$ occur in the equations:

Step 1. Standardize equations: Transform all equations into the form

$$a_1B_1 + \cdots + a_mB_m = 0,$$

where the B_i are different variables and the a_i are integers.

Step 2. Reduce the number of equations, using Lemma 11, applied to the torsion-free abelian group $\mathbb{Z}^{2^h \times 2^h}$.

Step 3. If a variable A of height h occurs in the equations, and the rule for A has the form $A \to B + C$, then replace every occurrence of A in the equations by $B + C$.

Step 4. If none of steps 1–3 applies to the equations, then only rules of the form

$$A \to \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \tag{3}$$

are applicable to a variable A (of height h) occurring in the equations. Applying all possible rules of this form for the current height results in a set of equations where all variables are (2×2)-matrices over variables of height $h - 1$ (like the right-hand side of (3)). Hence, every equation can be decomposed into 4 equations, where all variables have height $h - 1$.

If the height of all variables is finally 0 and none of steps 1–3 applies to the equations, then only rules of the form $A \to a$ with $a \in \mathbb{Z}$ are applicable. In this case, we replace all variables by the corresponding integers, and check whether all resulting equations are valid or not. If all equations hold, then the input equation holds, i.e., $\mathsf{val}(A_1) = \mathsf{val}(A_2)$. Otherwise, if at least one equation is not valid, then $\mathsf{val}(A_1) \neq \mathsf{val}(A_2)$.

The number of variables in the equations is bounded by the number of variables of G. An upper bound on the absolute value of the coefficients in the equations is $2^{|G|}$, since only iterated addition can be performed to increase the coefficients. Hence, $O(|G|)$ bits suffice for storing a coefficient. Lemma 11 shows that the number of equations after step 2 above is at most $|G|$ (the bound for the number of different variables).

For the case $S = \mathbb{Z}_k$ the same procedure works, we only have to use Lemma 12 instead of Lemma 11. □

Corollary 14 *Let M be a finitely generated cancellative commutative monoid. Given a +MTDD G over M and two variables A_1 and A_2 of G of the same height, one can check* $\mathsf{val}(A_1) = \mathsf{val}(A_2)$ *in time polynomial in* $|G|$.

Proof A cancellative commutative monoid M embeds into its Grothendieck group A (see e.g. [42]), which is the quotient of $M \times M$ by the congruence defined by $(a, b) \equiv (c, d)$ if and only if $a + d = c + b$ in M.[3] This is an abelian group, which is moreover finitely generated if M is finitely generated. Hence, the result follows from Theorem 14. □

Let us now consider non-cancellative commutative monoids:

Theorem 15 *Let M be a non-cancellative finitely generated commutative monoid. It is* coNP*-complete to check* $\mathsf{val}(A_1) = \mathsf{val}(A_2)$ *for a given +MTDD G over M and two variables A_1 and A_2 of G of the same height.*

Proof We start with the upper bound. Let $\{a_1, \dots, a_k\}$ be a finite generating set of M. Let G be a +MTDD over M and let A_1 and A_2 two variables of G of height h. It suffices to check in polynomial time for two given indices $1 \le i, j \le 2^h$ whether $\mathsf{val}(A_1)_{i,j} = \mathsf{val}(A_2)_{i,j}$. Then, this test has to be done for all (exponentially many) $1 \le i, j \le 2^h$, which corresponds to the universal quantifier in coNP.

From $1 \le i, j \le 2^h$ we can compute +-circuits for the matrix entries $\mathsf{val}(A_1)_{i,j}$ and $\mathsf{val}(A_2)_{i,j}$, see Proposition 8(3). From these circuits we can compute numbers $n_1, \dots, n_k, m_1, \dots, m_k \in \mathbb{N}$ in binary representation such that $\mathsf{val}(A_1)_{i,j} = n_1a_1 + \cdots + n_ka_k$ and $\mathsf{val}(A_2)_{i,j} = m_1a_1 + \cdots + m_ka_k$. Now we can use the following result from [35]: There is a semilinear subset $S \subseteq \mathbb{N}^{2k}$ (depending only on our fixed commutative monoid M) such that for all $x_1, \dots, x_k, y_1, \dots, y_k \in \mathbb{N}$ we have: $x_1a_1 + \cdots + x_ka_k = y_1a_1 + \cdots + y_ka_k$ if and only if $(x_1, \dots, x_k, y_1, \dots, y_k) \in S$. Hence, we have to check, whether $v := (n_1, \dots, n_k, m_1, \dots, m_k) \in S$. The semilinear set S is a finite union of linear sets. Hence, we can assume that S is linear itself. Let

$$S = \{v_0 + \lambda_1 v_1 + \cdots + \lambda_l v_l \mid \lambda_1, \dots, \lambda_l \in \mathbb{N}\},$$

where $v_0, \dots, v_l \in \mathbb{N}^{2k}$. Thus, we have to check, whether there exist $\lambda_1, \dots, \lambda_l \in \mathbb{N}$ such that $v = v_0 + \lambda_1 v_1 + \cdots \lambda_l v_l$. This is an instance of integer programming in the fixed dimension $2k$, which can be solved in polynomial time [24].

Let us now prove coNP-hardness. We do this for 1-dimensional +MTDDs that produce vectors instead of matrices. We fix elements $x, y, z \in M$ such that $x \neq y$ but $x + z = y + z$. These elements exist since M is not cancellative. We use an encoding of 3SAT from [4]. Take a 3CNF formula $C = \bigwedge_{i=1}^m C_i$ over n propositional variables $x_1, \dots, x_n$, and let $C_i = (\alpha_{j_1} \vee \alpha_{j_2} \vee \alpha_{j_3})$, where $1 \le j_1 < j_2 < j_3 \le n$ and every α_{j_k} is either x_{j_k} or $\neg x_{j_k}$. With this clause we associate a 1-dimensional MTDD G_i as follows: The variables are $A_0, \dots, A_n$ and $B_0, \dots, B_{n-1}$, where B_i produces the vector of length 2^i with all entries

[3] This construction generalizes the construction of $\mathbb{Z}$ from $\mathbb{N}$.

equal to 0 (which corresponds to the truth value true, whereas $z \in M$ corresponds to the truth value false). For the variables $A_0, \ldots, A_n$ we add the following rules: For every $1 \leq j \leq n$ with $j \notin \{j_1, j_2, j_3\}$ we take the rule $A_j \to (A_{j-1}, A_{j-1})$. For every $j \in \{j_1, j_2, j_3\}$ such that $\alpha_j = x_j$ ($\alpha_j = \neg x_j$, respectively) we take the rule

$$A_j \to (A_{j-1}, B_{j-1}) \ \ (A_j \to (B_{j-1}, A_{j-1}), \text{ respectively}).$$

Finally add the rule $A_0 \to z$ (more formally, we have to use a fixed size +MTDD to produce z from the monoid generators) and let A_n be the start variable of G_i. Then $\mathsf{val}(G_i)$ is a vector of length 2^n that encodes all truth values of clause C_i in the following sense: The j-th lexicographic truth assignment to the propositional variables $x_1, \ldots, x_n$ makes clause C_i true (false, respectively) if the j-th component of $\mathsf{val}(G_i)$ is 0 (z, respectively).

Next let G (H, respectively) be the 1-dimensional MTDD that produces the vector consisting of 2^n many x-entries (y-entries, respectively). Then,

$$\mathsf{val}(G) + \mathsf{val}(G_1) + \cdots + \mathsf{val}(G_m) = \mathsf{val}(H) + \mathsf{val}(G_1) + \cdots + \mathsf{val}(G_m)$$

if and only if for every $1 \leq j \leq 2^n$ there exists $1 \leq i \leq m$ such that the j-th component of $\mathsf{val}(G_i)$ is z. The latter holds if and only if for every truth assignment one of the clauses C_i is false, i.e., if and only if C is unsatisfiable. □

6 Computing determinants and matrix powers

In this section we present several completeness results for MTDDs over the rings $\mathbb{Z}$ and $\mathbb{Z}_n$ ($n \geq 2$). It turns out that over these rings, computing determinants, iterated matrix products, or matrix powers are infeasible for MTDD-represented input matrices, assuming standard assumptions from complexity theory. All completeness results in this section are formulated for MTDDs, but they remain valid if we add addition. In fact, all upper complexity bounds in this section even hold for matrices that are represented by circuits as defined in Section 4.2.

All hardness results in this section rely on the fact that the adjacency matrix of the configuration graph of a polynomial space bounded machine can be produced by a small MTDD (with terminal entries 0 and 1), see Section 6.2. This was also shown in [12, proof of Theorem 7] in the context of OBDDs. We will prove this fact using an automata theoretic framework that we introduce in Section 6.1. This framework will simplify the technical details in the proofs in Sections 6.3 and 6.4.

6.1 Layered automata and MTDDs

In the following we will use some standard notations concerning finite automata. A *layered DFA (deterministic finite automaton) of depth m* is an acyclic DFA A for which the state set Q is partitioned into $m+1$ layers $Q_0, \ldots, Q_m$ such that:

- Q_0 only contains the initial state q_0 of A.
- Q_m only contains two states, one of which is the unique final state of A.
- Every transition goes from layer Q_i to Q_{i+1} for some $0 \leq i < m$.
- For every state $q \in Q_i$ $(1 \leq i < m)$ and every input letter a there exists an a-labeled transition from q to a state from layer Q_{i+1}.

The *convolution* of a string $u = a_1 \cdots a_n \in \Sigma^*$ and a string $v = b_1 \cdots b_n \in \Gamma^*$ is the string $u \otimes v = (a_1, b_1) \cdots (a_n, b_n)$ over the alphabet $\Sigma \times \Gamma$. A layered DFA A of depth m with input alphabet $\{0,1\} \times \{0,1\}$ defines the directed graph $\mathcal{G}(A)$ with node set $\{0,1\}^m$ (all binary strings of length m) and an edge from $u \in \{0,1\}^m$ to $v \in \{0,1\}^m$ if and only if $u \otimes v \in L(A)$. So, A recognizes the edge relation of $\mathcal{G}(A)$. Layered DFAs over the paired alphabet $\{0,1\} \times \{0,1\}$ are basically the same as MTDDs over $\{0,1\}$ (or OBDDs with the interleaved variable ordering):

Lemma 16 *One can construct in logspace from a given layered DFA A over the paired alphabet $\{0,1\} \times \{0,1\}$ an MTDDs G over $\{0,1\}$ such that $\mathsf{val}(G)$ is the adjacency matrix of the graph $\mathcal{G}(A)$, and vice versa.*

Proof The variables of G are the states of the automaton A, and the start variable is the initial state q_0. Let $P_0, \ldots, P_k$ be the layers of A and let $P_k = \{p_0, p_1\}$, where p_1 is the final state of A. First, we add the rule $p_i \to i$ for $i \in \{0,1\}$ to G. Next, let $p \in P_i$ for some $i < k$ and let

$$p \xrightarrow{(a,b)} p_{a,b}$$

for $a, b \in \{0,1\}$ be the four outgoing transitions from state p. Then we add the rule

$$p \to \begin{pmatrix} p_{0,0} & p_{0,1} \\ p_{1,0} & p_{1,1} \end{pmatrix}$$

to G. The reverse transformation works similarly. □

6.2 Generating the configuration graph of a Turing machine by an MTDD

Let M be a nondeterministic Turing machine (NTM). Let Q be the set of states of M, and let Γ be the tape alphabet of M, where $Q \cap \Gamma = \emptyset$. As usual, configurations of M are encoded as words from $\Gamma^* Q \Gamma^*$. For two configurations $c_1, c_2 \in \Gamma^* Q \Gamma^*$ we write $c_1 \vdash_M c_2$ if M can move in one transition from configuration c_1 to configuration c_2. Let us fix an injective encoding $f_M : Q \cup \Gamma \to \{0,1\}^{k_M} \setminus 0^*$, which is extended to a homomorphism from $(Q \cup \Gamma)^*$ to $\{0,1\}^*$. Here, k_M is a large enough constant. We exclude words only consisting of 0's from the range of f_M for technical reasons. The following proposition makes use of the folklore fact (see e.g. the work on automatic structures [6]) that a Turing machine transition only locally modifies the current configuration and that this local modification can be recognized by a finite automaton. This locality is not destroyed by an application of the coding function f_M:

Lemma 17 *Let M be a fixed NTM. For $m \in \mathbb{N}$, one can compute in space $O(\log m)$ a layered DFA $A(M, m)$ of depth $k_M(m+1)$ over the paired alphabet $\{0,1\} \times \{0,1\}$ such that*

$$L(A(M,m)) = \{f_M(c_1) \otimes f_M(c_2) \mid c_1, c_2 \in \Gamma^* Q \Gamma^*,\\ |c_1| = |c_2| = m+1, c_1 \vdash_M c_2\}.$$

Proof Due to the local nature of Turing machines, there exists a fixed complete DFA $A(M)$ over the alphabet $\{(0,0),(0,1),(1,0),(1,1)\}$ such that

$$L(A(M)) = \{f_M(c_1) \otimes f_M(c_2) \mid c_1, c_2 \in \Gamma^* Q \Gamma^*, |c_1| = |c_2|, c_1 \vdash_M c_2\}.$$

Using the classical product construction, we intersect this automaton with a layered DFA of depth $k_M(m+1)$ for the language

$$\{0,1\}^{k_M(m+1)} \otimes \{0,1\}^{k_M(m+1)}.$$

Such an automaton can be constructed in logspace. The resulting DFA is layered and recognized the desired language. □

For the layered DFA $A(M,m)$ from Lemma 17, the graph $\mathcal{G}(A(M,m))$ is the configuration graph of M on configurations of tape length m. With Lemma 16 we can compute in space $\log m$ an MTDD for the adjacency matrix of this configuration graph.

6.3 Hardness of the determinant for MTDDs

Recall that the determinant of a matrix $A = (a_{i,j})_{1 \le i,j \le n}$ (over any ring) can be computed as follows, where S_n denotes the set of all permutations on $\{1,\dots,n\}$:

$$\det(A) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \cdot \prod_{i=1}^{n} a_{i,\sigma(i)}.$$

Here, $\operatorname{sgn}(\sigma)$ denotes the signum of the permutation σ, which is 1 (-1, respectively) if σ is a product of an even (odd, respectively) number of transpositions. If A is the adjacency matrix of a directed graph $\mathcal{G}$, then we can compute $\det(A)$ by taking the sum over all cycle covers of $\mathcal{G}$ (a cycle cover of $\mathcal{G}$ is a subset of the edges of $\mathcal{G}$ such that the corresponding subgraph is a disjoint union of directed cycles), where each cycle cover contributes to the sum by the signum of the corresponding permutation. If the underlying ring R is commutative, then A is invertible if and only if $\det(A)$ is invertible in R. The value $\det(\mathsf{val}(G))$ for an MTDD G over $\mathbb{N}$ may be of doubly exponential size (and hence needs exponentially many bits): The diagonal $(2^n \times 2^n)$-matrix with 2's on the diagonal has determinant 2^{2^n}.

By the next theorem, computing the determinant of an MTDD-represented matrix is indeed difficult. To prove this result we use a reduction of Toda showing that computing the determinant of an explicitly given integer matrix is GapL-complete [36] (which in turn is based on Valiant's classical construction for the universality of the determinant [39]). We apply this reduction to configuration graphs of polynomial space bounded Turing machines, whose adjacency matrices can be produced by small MTDDs.

Theorem 18 *The following holds for every ring $S \in \{\mathbb{Z}\} \cup \{\mathbb{Z}_n \mid n \geq 2\}$:*

(1) The set $\{G \mid G \text{ is an MTDD over } S, \det(\mathsf{val}(G)) = 0\}$ is PSPACE-complete.

(2) The function $G \mapsto \det(\mathsf{val}(G))$ with G an MTDD over $\mathbb{Z}$ is $\mathsf{GapPSPACE}$-complete.

Proof Let us start with the upper bounds. Membership in PSPACE in statement (1) can be shown as follows: Since the determinant of an explicitly given integer matrix can be computed in $\mathsf{FSPACE}(\log^2(n))$, one can check in $\mathsf{DSPACE}(\log^2(n))$ whether the determinant of an explicitly given integer matrix is zero. Moreover, from a given MTDD G we can compute the matrix $\mathsf{val}(G)$ in polynomial space. For this, it suffices to compute for G and given positions i, j the entry $\mathsf{val}(G)_{i,j}$ in PSPACE; then we can iterate over all (exponentially many) matrix positions (i, j). Actually, a specific matrix entry $\mathsf{val}(G)_{i,j}$ can be even computed in polynomial time by Proposition 8(3). Membership in PSPACE for MTDDs follows from Lemma 1. Note that the same argument even applies for matrices that are represented by boolean circuits in the sense of Definition 9 as well as Definition 10 from Section 4.2.

The upper bounds in (2) can be shown in the same way using Lemma 2 and the fact that a function $f : \{0,1\}^* \to \mathbb{Z}$ belongs to $\mathsf{GapPSPACE}$ if and only if the mapping $x \mapsto \mathsf{bin}(f(x))$ belongs to $\mathsf{FPSPACE}$.

Let us now prove the lower bounds. We start with (1). Let us take a deterministic polynomial space bounded Turing machine M. Let q_0 be the initial state of M and q_f the unique accepting state. Let $\Box$ be the blank symbol. We can assume that M is non-looping in the sense that there does not exist a configuration c such that $c \vdash_M^+ c$. This property can be ensured by adding a binary counter to M that is decremented during each transition of the original machine. Moreover, we can assume that every accepting computation path of M has odd length (i.e., an odd number of transitions), and that every tape cell contains $\Box$ as soon as M enters the accepting state q_f. Let $p(n)$ (a polynomial) be the space bound of M and let x be an input for M of length n. Moreover, let $m = p(n)$ and $k = k_M(m+1)$. By Lemma 17 we can compute in space $O(\log m) = O(\log n)$ a layered DFA $A(M, m)$ of depth k such that

$$L(A(M,m)) = \{f_M(c_1) \otimes f_M(c_2) \mid c_1, c_2 \in \Gamma^* Q \Gamma^*, \\ |c_1| = |c_2| = m+1, c_1 \vdash_M c_2\}.$$

Let $w_0 = f_M(q_0 x \Box^{m-n})$ ($w_f = f_M(q_f \Box^m)$, respectively) be the encoding of the initial (accepting, respectively) configuration. Recall that we assume that

0^k does not belong to $f_M(\Gamma^*Q\Gamma^*)$. By taking the direct product of $A(M,m)$ with a layered DFA for the language

$$K = \{0^k \otimes w_0, w_f \otimes 0^k\} \cup \{w \otimes w \mid w \in \{0,1\}^k \setminus 0^*\}$$

(which can be computed in space $\log m$), we obtain a layered DFA $A(M,x)$ with $L(A(M,x)) = L(A(M,m)) \cup K$. Let $\mathcal{G}(M,x)$ be the directed graph $\mathcal{G}(A(M,x))$ defined by the DFA $A(M,x)$. Its node set is $\{0,1\}^k$ and there is an edge from v to w if and only if $v \otimes w \in L(A(M,x))$. Let $\mathrm{adj}(M,x)$ be the adjacency matrix of $\mathcal{G}(M,x)$. We compute $\det(\mathrm{adj}(M,x))$ by considering cycle covers of the graph $\mathcal{G}(M,x)$. Note that node 0^k lies on a directed cycle if and only if there is a path from w_0 to w_f in $\mathcal{G}(M,x)$. Moreover, since M is non-looping, every cycle cover of $\mathcal{G}(M,x)$ consists of a path from w_0 to w_f together with the two edges $(w_f, 0^k)$ and $(0^k, w_0)$ (such a cycle has odd length and hence is a product of an even number of transpositions) together with loops on the remaining nodes. It follows that in $\mathbb{Z}$, $\det(\mathrm{adj}(M,x))$ is equal to the number of paths from w_0 to w_f in $\mathcal{G}(M,x)$. But this number is equal to the number of accepting computations of the machine M on input x, which is either 0 or 1 (since M is deterministic). By Lemma 16 applied to the DFA $A(M,x)$, we obtain in logspace an MTDD G (with integer entries 0 and 1 only) such that $\mathsf{val}(G) = \mathrm{adj}(M,x)$. This shows the lower bound in (1).

Let us finally prove the lower bound in (2). Let us take two polynomial space bounded nondeterministic Turing machines M_1 and M_2 with the same input alphabet. We can also assume that M_1 and M_2 have the same state set Q and tape alphabet Γ. In particular, we can assume that $k_{M_1} = k_{M_2}$. Let $f = f_{M_1} = f_{M_2}$ be the binary coding mapping for $Q \cup \Gamma$. Let q_0 be the initial state of M_1 and M_2 and q_f the unique accepting state of M_1 and M_2. We make the same assumptions that we have made for M in the lower bound proof for statement (1). We can also assume that the polynomial $p(n)$ is a space bound for M_1 as well as M_2.

Let x be an input for M_1 and M_2 of length n, and let $m = p(n)$, $k = k_{M_1}(m+1) = k_{M_2}(m+1)$. By Lemma 17 we can construct in space $O(\log m) = O(\log n)$ layered DFAs $A(M_1,m)$ and $A(M_2,m)$ of depth k such that

$$L(A(M_i,m)) = \{f(c_1) \otimes f(c_2) \mid c_1, c_2 \in \Gamma^*Q\Gamma^*, |c_1| = |c_2| = m+1, c_1 \vdash_{M_i} c_2\}.$$

Let $w_0 = f(q_0 x \Box^{m-n})$ be the encoding of the initial configuration, and let $w_f = f(q_f \Box^m)$ be the encoding of the unique accepting configuration. Recall that we assumed that 0^k does not belong to $f(\Gamma^*Q\Gamma^*)$.

From the layered DFAs $A(M_1,m)$ and $A(M_2,m)$ we now construct a layered DFA $A(M_1,M_2,m)$ of depth $k+1$ such that

$$\begin{aligned} L(A(M_1,M_2,m)) = {} & \{0u \otimes 0v \mid u \otimes v \in L(A(M_1,m))\} \ \cup \\ & \{1u \otimes 1v \mid u \otimes v \in L(A(M_2,m))\}. \end{aligned}$$

For this we basically take the disjoint union of $A(M_1, m)$ and $A(M_2, m)$. By taking the product of $A(M_1, M_2, m)$ with a layered DFA for the language

$$K = \{0^{k+1} \otimes 0w_0, 0w_f \otimes 0^{k+1}, 0^{k+1} \otimes 1w_0, 1w_f \otimes 10^k, 10^k \otimes 0^{k+1}\} \cup \\ \{w \otimes w \mid w \in \{0,1\}^{k+1} \setminus 0^*\}$$

(which can be easily constructed in space $O(\log k) = O(\log n)$), we obtain a layered DFA $A(M_1, M_2, x)$ with

$$L(A(M_1, M_2, x)) = L(A(M_1, M_2, m)) \cup K.$$

Let $\mathcal{G}(M_1, M_2, x)$ be the directed graph $\mathcal{G}(A(M_1, M_2, x))$ defined by the layered DFA $A(M_1, M_2, x)$. This graph consists of the disjoint union of the two graphs $\mathcal{G}(M_1, m) := \mathcal{G}(A(M_1, m))$ and $\mathcal{G}(M_2, m) := \mathcal{G}(A(M_2, m))$ (basically the configurations graphs of M_1 and M_2 on configurations of tape length m) together with two nodes 0^{k+1} and 10^k and the following edges:

- Edges from 0^{k+1} to $0w_0$ and $1w_0$ (the copies of the initial configuration in the graphs $\mathcal{G}(M_1, m)$ and $\mathcal{G}(M_2, m)$).
- An edge from $0w_f$ (the copy of the accepting configuration in $\mathcal{G}(M_1, m)$) back to 0^{k+1}.
- An edge from $1w_f$ (the copy of the accepting configuration in $\mathcal{G}(M_2, m)$) to 10^k.
- An edge from 10^k back to 0^{k+1}.
- Loops at all nodes except for 0^{k+1}.

Let $\mathrm{adj}(M_1, M_2, x)$ be the adjacency matrix of the directed graph $\mathcal{G}(M_1, M_2, x)$. Let us compute $\det(\mathrm{adj}(M_1, M_2, x))$ by considering cycle covers of the graph $\mathcal{G}(M_1, M_2, x)$. Note that node 0^{k+1} lies on a directed cycle if and only if there is a path from w_0 to w_f in $\mathcal{G}(M_1, x)$ or from w_0 to w_f in $\mathcal{G}(M_2, x)$. Moreover, since M_1 and M_2 are non-looping, every cycle cover of $\mathcal{G}(M_1, M_2, x)$ consists of loops together with either

- a path from node $0w_0$ to node $0w_f$ (in $\mathcal{G}(M_1, m)$) together with the two edges $(0^{k+1}, 0w_0)$ and $(0w_f, 0^{k+1})$ (every such cycle has odd length, and hence is a product of an even number of transpositions), or
- a path from node $1w_0$ to node $1w_f$ (in $\mathcal{G}(M_2, m)$) together with the three edges $(0^{k+1}, 1w_0)$, $(1w_f, 10^k)$, and $(10^k, 0^{k+1})$ (every such cycle has even length, and hence is a product of an odd number of transpositions).

It follows that $\det(\mathrm{adj}(M_1, M_2, x))$ is equal to the number of paths from $0w_0$ to $0w_f$ in $\mathcal{G}(M_1, m)$ minus the number of paths from $1w_0$ to $1w_f$ in $\mathcal{G}(M_2, m)$. But this number is equal to the number of accepting computations of the machine M_1 on input x minus the number of accepting computations of the machine M_2 on input x. □

Note that the determinant of a diagonal matrix is zero if and only if there is a zero-entry on the diagonal. This can be easily checked in polynomial time for a diagonal matrix produced by an MTDD. For +MTDDs (actually, for sums of several MTDD-represented matrices) we can show NP-completeness of this problem:

Theorem 19 *It is* NP*-complete to check* $\det(\mathsf{val}(G_1) + \cdots + \mathsf{val}(G_k)) = 0$ *for given MTDDs* $G_1, \ldots, G_k$ *that produce diagonal matrices of the same dimension* 2^n.

Proof Membership in NP is easy: Simply guess a position $1 \le i \le 2^n$, compute the values $n_j = \mathsf{val}(G_j)_{i,i}$ for $1 \le j \le k$ and check whether $n_1 + \cdots + n_k = 0$.

Our NP-hardness proof uses again the 3SAT encoding from [4] that we applied in the proof of Theorem 15. Take a boolean formula $C = \bigwedge_{i=1}^m C_i$, where every C_i is a disjunction of three literals. Assume that $x_1, \ldots, x_n$ are the boolean variables that occur in C. For each $1 \le i \le m$ let $w_i \in \{0,1\}^{2^n}$ be the bit vector of length 2^n, where the j-th entry of w_i $(1 \le j \le 2^n)$ is 1 if and only if the lexicographically j-th truth assignment to the variables $x_1, \ldots, x_n$ satisfies clause C_i. As in the proof of Theorem 15 one can construct from C_i in logspace a 1-dimensional MTDD that produces w_i. We can use the same construction in order to construct in logspace a (2-dimensional) MTDD G_i of height n such that $\mathsf{val}(G_i)$ is a diagonal matrix with the vector w_i on the diagonal. Here is the construction: Let $C_i = (\alpha_{j_1} \vee \alpha_{j_2} \vee \alpha_{j_3})$, where $1 \le j_1 < j_2 < j_3 \le n$ and every α_{j_k} is either x_{j_k} or $\neg x_{j_k}$. We take variables $A_0, \ldots, A_n$, $B_0, \ldots, B_{n-1}$, $Z_0, \ldots, Z_{n-1}$, where B_i produces the $(2^i \times 2^i)$-dimensional identity matrix and Z_i produces the $(2^i \times 2^i)$-dimensional zero matrix. For the variables $A_0, \ldots, A_n$ we add the following rules: For every $1 \le j \le n$ with $j \notin \{j_1, j_2, j_3\}$ take the rule

$$A_j \to \begin{pmatrix} A_{j-1} & Z_{j-1} \\ Z_{j-1} & A_{j-1} \end{pmatrix}.$$

For every $j \in \{j_1, j_2, j_3\}$ such that $\alpha_j = x_j$ take the rule

$$A_j \to \begin{pmatrix} A_{j-1} & Z_{j-1} \\ Z_{j-1} & B_{j-1} \end{pmatrix}.$$

For every $j \in \{j_1, j_2, j_3\}$ such that $\alpha_j = \neg x_j$ take the rule

$$A_j \to \begin{pmatrix} B_{j-1} & Z_{j-1} \\ Z_{j-1} & A_{j-1} \end{pmatrix}.$$

Finally we take the rule $A_0 \to 0$. Let A_n be the initial variable of G_i. Then, indeed, $\mathsf{val}(G_i)$ is a diagonal matrix with the bit vector w_i on the diagonal for $1 \le i \le m$. Let G_{m+1} be an MTDD such that $\mathsf{val}(G_{m+1}) = -mI_{2^n}$, where I_{2^n} is the $(2^n \times 2^n)$-dimensional identity matrix. Then $\mathsf{val}(G_1) + \cdots + \mathsf{val}(G_{m+1})$ is a diagonal matrix which has a zero on the diagonal (i.e., $\det(\mathsf{val}(G_1) + \cdots + \mathsf{val}(G_{m+1})) = 0$) if and only if the 3CNF formula C is satisfiable. □

6.4 Hardness of iterated multiplication and powering for MTDDs

Let us now discuss the complexity of iterated multiplication and powering. Computing a specific entry, say at position $(1,1)$, of the product of n explicitly given matrices over $\mathbb{Z}$ ($\mathbb{N}$, respectively) is known to be complete for GapL

(#L, respectively) [36]. Corresponding results hold for the computation of the $(1,1)$-entry of a matrix power A^n, where n is given in unary notation. Hence, the binary encodings of these numbers can be computed in $\mathsf{FSPACE}(\log^2(n))$. As usual, these problems become exponentially harder for matrices that are encoded by boolean circuits (see Section 4.2). Let us briefly discuss two scenarios. Recall the matrices $M_{C,1}$ (Definition 9) and $M_{C,2}$ (Definition 10) defined from a circuit C in Section 4.2.

Definition 20 For a tuple $\overline{C} = (C_1, \ldots, C_n)$ of boolean circuits we can define the matrix product $M_{\overline{C}} = \prod_{i=1}^{n} M_{C_i,1}$.

Lemma 21 *The function $\overline{C} \mapsto (M_{\overline{C}})_{1,1}$, where every matrix $M_{C_i,1}$ is over $\mathbb{N}$ ($\mathbb{Z}$, respectively), belongs to #P (GapP, respectively).*

Proof Let us first show the result for #P. Let $\overline{C} = (C_1, \ldots, C_n)$ and let $\overline{x}, \overline{y}$ be the input bits for the circuits C_i $(1 \le i \le n)$ where $m = |\overline{x}| = |\overline{y}|$. Let

$$M_{C_i,1} = (a^{(i)}_{j,k})_{1 \le j,k \le 2^m} \in \mathbb{N}^{2^m \times 2^m}$$

be the matrix defined by the circuit C_i. We have

$$\Big(\prod_{i=1}^{n} M_{C_i,1}\Big)_{1,1} = \sum_{i_1=1}^{2^m} \sum_{i_2=1}^{2^m} \cdots \sum_{i_{n-1}=1}^{2^m} a^{(1)}_{1,i_1} a^{(2)}_{i_1,i_2} \cdots a^{(n-1)}_{i_{n-2},i_{n-1}} a^{(n)}_{i_{n-1},1}. \tag{4}$$

We have to come up with a nondeterministic polynomial time Turing machine M that has that many accepting computation paths on input $(C_1, \ldots, C_n)$. Using $(n-1) \cdot m$ binary branchings, the machine M can produce an arbitrary tuple $(i_1, \ldots, i_{n-1})$, where the numbers $1 \le i_1, \ldots, i_{n-1} \le 2^m$ are written down in binary notation. By evaluating the circuits C_i on the right input bits, we can compute in deterministic polynomial time the binary codings of all natural numbers $a^{(1)}_{1,i_1}, a^{(2)}_{i_1,i_2}, \ldots, a^{(n-1)}_{i_{n-2},i_{n-1}}, a^{(n)}_{i_{n-1},1}$. Then we compute the product a of these numbers again deterministically in polynomial time. If $a = 0$ then we reject on the current computation path (this corresponds to a 0 in the multiple sum (4)). Otherwise, using the binary coding of $a > 0$ the machine branches $\lceil \log a \rceil$ many times in order to produce a many accepting computation paths.

For the statement concerning GapP one can argue similarly. We have to come up with two polynomial space bounded Turing machines such that $\big(\prod_{i=1}^{m} M_{C_i,1}\big)_{1,1}$ is equal to the number of accepting computations of the first machine minus the number of accepting computations of the second machine. These two machines work as above, but the first (second, respectively) machine only produces $a = a^{(1)}_{1,i_1}, a^{(2)}_{i_1,i_2}, \ldots, a^{(n-1)}_{i_{n-2},i_{n-1}}, a^{(n)}_{i_{n-1},1}$ many accepting computation paths if $a > 0$ ($a < 0$, respectively). □

An alternative and more succinct way of specifying a long matrix product uses a single circuit:

Definition 22 A boolean circuit $C(\overline{w}, \overline{x}, \overline{y}, \overline{z})$ with $k = |\overline{w}|$, $m = |\overline{x}|$, and $n = |\overline{y}| = |\overline{z}|$ encodes a sequence of 2^k many $(2^n \times 2^n)$-matrices: For every bit vector $\overline{a} \in \{0,1\}^k$, define the circuit $C_{\overline{a}} = C(\overline{a}, \overline{x}, \overline{y}, \overline{z})$ and the matrix $M_{\overline{a}} = M_{C_{\overline{a}},2}$ (see Definition 10). Finally, let $M_C = \prod_{\overline{a} \in \{0,1\}^k} M_{\overline{a}}$ be the product of all these matrices.

Lemma 23 *The function* $C(\overline{w}, \overline{x}, \overline{y}, \overline{z}) \mapsto M_C$ *belongs to* $\mathsf{FPSPACE}$.

Proof The lemma follows from Lemma 2 and the following two facts: (i) From the circuit $C(\overline{w}, \overline{x}, \overline{y}, \overline{z})$ one can compute the tuple of matrices $(M_{C_{\overline{a}},2})_{\overline{a} \in \{0,1\}^k}$ in polynomial space (simply iterate over all valuations for the boolean variables $\overline{w}, \overline{x}, \overline{y}, \overline{z}$), and (ii) computing an iterated matrix product of explicitly given matrices can be done in $\mathsf{FSPACE}(\log^2(n))$. □

Lemmas 21 and 23 yield the upper complexity bounds in the following theorem.

Theorem 24 *The following hold:*

(1) The function $(G, n) \mapsto (\mathsf{val}(G)^n)_{1,1}$ *with* G *an MTDD over* $\mathbb{N}$ ($\mathbb{Z}$, *respectively) and* n *a unary encoded number is complete for* $\#\mathsf{P}$ (GapP, *respectively).*

(2) The function $(G, n) \mapsto (\mathsf{val}(G)^n)_{1,1}$ *with* G *an MTDD over* $\mathbb{N}$ ($\mathbb{Z}$, *respectively) and* n *a binary encoded number is* $\#\mathsf{PSPACE}$-*complete* ($\mathsf{GapPSPACE}$-*complete, respectively).*

Proof The upper bound in (1) follows from Lemma 21: Simply convert the MTDD G into an equivalent circuit C and apply Lemma 21 to the circuit tuple $(C, \ldots, C)$ with n copies of C. Similarly, the upper bound in (2) follows from Lemma 23: From (G, n) (with n in binary encoding) one can compute a single boolean circuit C such that M_C (as defined in Definition 22) is the matrix $\mathsf{val}(G)^n$.

It remains to prove the lower bound, for which we use again succinct versions of Toda's techniques from [36], similar to the proof of Theorem 18.

Let us start with the statements concerning $\#\mathsf{P}$ and $\#\mathsf{PSPACE}$. We start with (1). Let M be a fixed nondeterministic polynomial time Turing machine. One can assume that all maximal computations of M on an input x of length n have length $p(n)$ for some polynomial p. Let x be an input for M of length n, and let $m = p(n)$ and $k = k_M(m+1)$ (see Section 6.3). We now apply the construction from the proof of Lemma 17 to M and m. We obtain a layered DFA $A(M, m)$ such that

$$L(A(M,m)) = \{f_M(c_1) \otimes f_M(c_2) \mid c_1, c_2 \in \Gamma^* Q \Gamma^*, \\ |c_1| = |c_2| = m+1, c_1 \vdash_M c_2\}.$$

Let $w_0 = f_M(q_0 x \Box^{m-n})$ be the encoding of the initial configuration, and $w_f = f_M(q_f \Box^m)$ be the encoding of the unique accepting configuration. Recall that 0^k does not belong to $f_M(\Gamma^* Q \Gamma^*)$. As in the proof of Theorem 18 we obtain a layered DFA $A(M, x)$ such that

$$L(A(M,x)) = L(A(M,m)) \cup \{0^k \otimes w_0, w_f \otimes 0^k\}.$$

Let $\mathcal{G}(M,x)$ be the directed graph $\mathcal{G}(A(M,x))$, whose node set is $\{0,1\}^k$ and there is an edge from v to w if and only if $v \otimes w \in L(A(M,x))$. Let $\mathrm{adj}(M,x)$ be the adjacency matrix of $\mathcal{G}(M,x)$. As in the proof of Theorem 18 we obtain an MTDD G such that $\mathsf{val}(G) = \mathrm{adj}(M,x)$.

Then the number of accepting computations of the machine M on input x is equal to the number of paths of length $p(n)+2$ in the graph $\mathcal{G}(M,x)$ from node 0^k back to node 0^k. This number is equal to $(\mathsf{val}(G)^{p(n)+2})_{1,1}$.

The #PSPACE-hardness in point (2) of the theorem is proven in the same way. For a nondeterministic polynomial space bounded Turing-machine one can assume that all maximal computations of M on an input x of length n have length $2^{p(n)}$ for some polynomial p. Hence, we only have to replace the number $p(n)+2$ in the above proof by $2^{p(n)}+2$.

Let us now turn to the lower bounds concerning GapP and $\mathsf{GapPSPACE}$ in the theorem. The proofs are very similar to the corresponding proofs for #P and #PSPACE, respectively. We only consider (2). We have to come up with an MTDD over $\{0,1,-1\}$. Such an MTDD corresponds to a layered DFA, where the last layer contains three states, corresponding to the three possible matrix entries 0, 1, and -1. Now, take two polynomial space bounded Turing machines M_1 and M_2 (with the same input alphabet), such that all accepting computations of M_1 and M_2 on an input of length n have length $2^{p(n)}$. Moreover, let x be an input for M_1 and M_2 of length n and let $m = p(n)$. Define the two graphs $\mathcal{G}(M_1,m)$ and $\mathcal{G}(M_2,m)$ (the configuration graphs of M_1 and M_2, respectively, on configurations of length m) as in the lower bound proof for (2) from Theorem 18.

We have to come up with a layered DFA (with three nodes in the last layer) that defines the following $\{1,-1\}$-labeled directed graph $\mathcal{G}$:

- $\mathcal{G}$ consists of a disjoint copy of $\mathcal{G}(M_1,m)$ and $\mathcal{G}(M_2,m)$ (all edges are labelled with 1) together with an additional node s.
- There is a 1-labeled edge from node s to the copy of the initial configuration of M_1 in $\mathcal{G}(M_1,m)$.
- There is a -1-labeled edge from node s to the copy of the initial configuration of M_2 in $\mathcal{G}(M_2,m)$.
- There are 1-labeled edges from the copies of the unique accepting configurations in M_1 and M_2, respectively, back to node s.

Analogously to the construction in the proof of (2) from Theorem 18 we can construct such a layered DFA. For the MTDD G over $\{0,1,-1\}$ corresponding to this layered DFA, $(\mathsf{val}(G)^{2^{p(m)}+2})_{1,1}$ is equal to the number of accepting computations of M_1 on input x minus the number of accepting computations of M_2 on input x. □

By Theorem 24, there is no polynomial time algorithm that computes for a given MTDD G and a unary number n a boolean circuit (or even a +MTDD) for the power $\mathsf{val}(G)^n$, unless #$\mathsf{P} = \mathsf{FP}$.

By [36] and Theorem 24, the complexity of computing a specific entry of a matrix power A^n covers three different counting classes, depending on the representation of the matrix A and the exponent n (let us assume that A is a matrix over $\mathbb{N}$):

- #L-complete, if A is given explicitly and n is given unary.
- #P-complete, if A is given by an MTDD and n is given unary.
- #PSPACE-complete, if A is given by an MTDD and n is given binary.

Let us also mention that in [8,15,31] the complexity of evaluating iterated matrix products and matrix powers in a fixed dimension is studied. It turns out that multiplying a sequence of $(d \times d)$-matrices over $\mathbb{Z}$ in the fixed dimension $d \geq 3$ is complete for the class GapNC^1 (the counting version of the circuit complexity class NC^1) [8]. It is open whether the same problem for matrices over $\mathbb{N}$ is complete for $\#\mathsf{NC}^1$. Moreover, the case $d = 2$ is open too. Matrix powers for matrices in a fixed dimension can be computed in TC^0 if the exponent is represented in unary notation [1,31] (the circuits from [1] are $\mathsf{DLOGTIME}$-uniform). The same problem for a binary encoded exponent belongs to the level $\mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}^{\mathsf{PP}}}}$ of the counting hierarchy [1]. Finally, multiplying a sequence of $(d \times d)$-matrices that is given succinctly by a boolean circuit captures the class $\mathsf{FPSPACE}$ for any $d \geq 3$ [15].

For the problem, whether a power of an MTDD-encoded matrix is zero (a variant of the classical mortality problem) we can finally show the following:

Theorem 25 *It is* coNP*-complete (*PSPACE*-complete, respectively) to check whether* $\mathsf{val}(G)^m$ *is the zero matrix for a given MTDD* G *over* $\mathbb{N}$ *and a unary (binary, respectively) encoded number* m.

Proof The PSPACE-bound for a binary encoded number m follows from the fact that the matrix $\mathsf{val}(G)^m$ can be computed from (G, m) in polynomial space, see Lemma 23 (the upper bound also holds if G is an MTDD over $\mathbb{Z}$).

Now assume that m is unary encoded. Let $\mathsf{val}(G) = (a_{i,j})_{1 \leq i,j \leq 2^h}$, where h is the height of G. Then $\mathsf{val}(G)^m$ is the zero matrix if and only if every sequence $a_{i_0,i_1}, a_{i_1,i_2}, \ldots, a_{i_{m-1},i_m}$ of matrix entries contains a zero. This is a coNP-property. Note that this argument needs the fact that all matrix entries $a_{i,j}$ of $\mathsf{val}(G)$ are non-negative.

For the lower bound, take the construction from the proof of the lower bound from point (1) of Theorem 24. Recall that $p(n)$ was the time bound of the machine M. We assumed that all maximal computation paths for an input of length n have length exactly $p(n)$. Let $m = p(n)$. We can modify the Turing machine M in such a way that the graph $\mathcal{G}(M, m)$ (the configuration graph of M on configurations of tape length m) does not have directed paths of length larger than m (e.g., by splitting the tape of M into two tracks and incrementing a unary counter on the second track). This means that in the graph $\mathcal{G}(M, x)$ there is a path of length $m + 2$ if and only if x is accepted by M. Thus, x is accepted by M if and only if $\mathsf{val}(G)^{p(n)+2}$ is not the zero matrix. The statement concerning PSPACE-completeness is proven in the same way (we just have to ensure by adding a binary counter on the second track that the graph $\mathcal{G}(M, m)$ does not have directed paths of length larger than $2^{p(n)}$). □

For an MTDD G over $\mathbb{Z}$ and a unary encoded number m, checking whether $\mathsf{val}(G)^m$ is the zero matrix seems to be harder due to possible cancellations.

Here is a more direct proof for the coNP-hardness statement in Theorem 25, which uses a reduction from the complement of 3SAT.

Alternative proof of Theorem 25. Let $C = \bigwedge_{i=1}^{m} C_i$ be a 3CNF formula. In the proof of Theorem 19 we constructed MTDDs $G_1, \ldots, G_m$ such that $\mathsf{val}(G_i)$ is the diagonal matrix, where the diagonal is the bit vector of all truth values of the clause C_i, taken in lexicographic order. From the MTDDs $G_1, \ldots, G_m$ we easily obtain an MTDD G such that

$$\mathsf{val}(G) = \begin{pmatrix} 0 & \mathsf{val}(G_1) & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \mathsf{val}(G_2) & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \mathsf{val}(G_3) & \cdots & 0 & 0 \\ & & & \vdots & & & \\ 0 & 0 & 0 & 0 & \cdots & \mathsf{val}(G_{m-1}) & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & \mathsf{val}(G_m) \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 \end{pmatrix}.$$

Here, we have to assume that $m+1$ is a power of two, which can be enforced by adding dummy clauses. Since the matrices $\mathsf{val}(G_i)$ commute (they are diagonal matrices) and are idempotent (since all diagonal values are 0 or 1), the matrix $\mathsf{val}(G)^m$ contains only 0-blocks except for the top right-most block, which is $\prod_{i=1}^{m} \mathsf{val}(G_i)$. Note that $\prod_{i=1}^{m} \mathsf{val}(G_i)$ is the diagonal matrix, where the diagonal is the bit vector of all truth values of the formula C. Thus, $\mathsf{val}(G)^m$ is the zero matrix if and only if C is unsatisfiable. □

7 Conclusion and future work

We studied algorithmic problems on matrices that are given by multi-terminal decision diagrams enriched by the operation of matrix addition. Several important matrix problems can be solved in polynomial time for this representation, e.g., equality checking, computing matrix entries, matrix multiplication, computing the trace, etc. On the other hand, computing determinants, matrix powers, and iterated matrix products are computationally hard. For further research, it should be investigated whether the polynomial time problems, like equality test, belong to NC.

One might also consider MTDDs with variables as entries. Theorem 24 and the results of [29] suggest that the algebraic complexity class $\mathsf{VPSPACE}$ can be characterized by exponential powers of MTDD-represented matrices with variable entries. In [29], the class $\mathsf{VPSPACE}$ is characterized in terms of succinct branching programs or, alternatively, exponential powers of circuit-represented matrices.

References

1. E. Allender, N. Balaji, and S. Datta. Low-depth uniform threshold circuits and the bit-complexity of straight line programs. In *Proceedings of MFCS 2014*, *LNCS 8635*, 13–24. Springer, 2014.
2. C. Àlvarez and B. Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107:3–30, 1993.
3. M. Beaudry and M. Holzer. The complexity of tensor circuit evaluation. *Computational Complexity*, 16(1):60–111, 2007.
4. P. Berman, M. Karpinski, L. L. Larmore, W. Plandowski, and W. Rytter. On the complexity of pattern matching for highly compressed two-dimensional texts. *Journal of Computer and System Sciences*, 65:332–350, 2002.
5. A. Bertoni, C. Choffrut, and R. Radicioni. Literal shuffle of compressed words. In *Proceedings of IFIP TCS 2008*, volume 273 of *IFIP*, 87–100. Springer, 2008.
6. A. Blumensath and E. Grädel. Finite presentations of infinite structures: Automata and interpretations. *Theory of Computing Systems*, 37:641–674, 2004.
7. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.
8. H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. *Journal of Computer and System Sciences*, 57(2):200–212, 1998.
9. S. A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2–22, 1985.
10. C. Damm, M. Holzer, and P. McKenzie. The complexity of tensor calculus. *Computational Complexity*, 11(1-2):54–89, 2002.
11. D. Eppstein, M. T. Goodrich, and J. Z. Sun. Skip quadtrees: Dynamic data structures for multidimensional point sets. *International Journal of Computational Geometry & Applications*, 18:131–160, 2008.
12. J. Feigenbaum, S. Kannan, M. Y. Vardi, and M. Viswanathan. The complexity of problems on graphs represented as obdds. *Chicago Journal of Theoretical Computer Science*, 1999.
13. H. Fujii, G. Ootomo, and C. Hori. Interleaving based variable ordering methods for ordered binary decision diagrams. In *Proceedings of ICCAD 1993*, 38–41. IEEE Computer Society, 1993.
14. M. Fujita, P. C. McGeer, and J. C.-Y. Yang. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10(2/3):149–169, 1997.
15. M. Galota and H. Vollmer. Functions computable in polynomial space. *Information and Computation*, 198(1):56–70, 2005.
16. H. Galperin and A. Wigderson. Succinct representations of graphs. *Information and Control*, 56:183–198, 1983.
17. V. Geffert, C. Mereghetti, and B. Palano. More concise representation of regular languages by automata and regular expressions. *Information and Computation*, 208(4):385–394, 2010.
18. B. Grenet, P. Koiran, and N. Portier. On the complexity of the multivariate resultant. *Journal of Complexity*, 29(2): 142–157, 2013.
19. M. Hayashida, P. Ruan, T. Akutsu. A quadsection algorithm for grammar-based image compression. *Integrated Computer-Aided Engineering*, 19(1): 23–38, 2012.
20. O. H. Ibarra and S. Moran. Probabilistic algorithms for deciding equivalence of straight-line programs. *Journal of the Association for Computing Machinery*, 30(1):217–228, 1983.
21. D. König and M. Lohrey. Evaluating matrix circuits. Technical report, arXiv.org, 2015. `http://arxiv.org/abs/1502.03540`, to appear in Proceedings of COCOON 2015.
22. R. E. Ladner. Polynomial space counting problems. *SIAM Journal on Computing*, 18:1087–1097, 1989.
23. T. Lengauer and K. W. Wagner. The correlation between the complexities of the nonhierarchical and hierarchical versions of graph problems. *Journal of Computer and System Sciences*, 44:63–93, 1992.

24. H. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.
25. M. Lohrey. Leaf languages and string compression. *Information and Computation*, 209:951–965, 2011.
26. M. Lohrey. Algorithmics on SLP-compressed strings: a survey. *Groups, Complexity, Cryptology*, 4:241-299, 2012.
27. M. Lohrey and S. Maneth. The complexity of tree automata and XPath on grammar-compressed trees. *Theoretical Computer Science*, 363(2):196–210, 2006.
28. M. Lohrey and C. Mathissen. Isomorphism of regular trees and words. *Information and Computation*, 224: 71–105, 2013.
29. G. Malod. Succinct algebraic branching programs characterizing non-uniform complexity classes. In *Proceedings of FCT 2011*, *LNCS 6914*, 205–216. Springer, 2011.
30. C. Meinel and T. Theobald. *Algorithms and Data Structures in VLSI Design: OBDD - Foundations and Applications*. Springer, 1998.
31. C. Mereghetti and B. Palano. Threshold circuits for iterated matrix product and powering. *Informatique Théorique et Applications*, 34(1):39–46, 2000.
32. W. Plandowski. Testing equivalence of morphisms in context-free languages. In *Proceedings of ESA 1994*, *LNCS 855*, 460–470. Springer, 1994.
33. H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.
34. A. Storjohann and T. Mulders. Fast algorithms for linear algebra modulo N. In *Proceedings of ESA 1998*, *LNCS 1461*, 139–150. Springer, 1998.
35. M. A. Taĭclin. Algorithmic problems for commutative semigroups. *Doklady Akademii Nauk SSSR*, 9(1):201–204, 1968.
36. S. Toda. Counting problems computationally equivalent to computing the determinant. Technical Report CSIM 91-07, Tokyo University of Electro-Communications, 1991.
37. S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20:865–877, 1991.
38. H. Torfah and M. Zimmermann. The complexity of counting models of linear-time temporal logic. In *Proceedings of FSTTCS 2014*, volume 29 of *LIPIcs*, pages 241–252. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014.
39. L. G. Valiant. Completeness classes in algebra. In *Proceedings of STOC 1979*, 249–261. ACM, 1979.
40. H. Veith. How to encode a logical structure by an OBDD. In *Proceedings of 13th Annual IEEE Conference on Computational Complexity*, 122–131. IEEE Computer Society, 1998.
41. I. Wegener. The size of reduced OBDD's and optimal read-once branching programs for almost all boolean functions. *IEEE Transactions on Computers*, 43(11):1262–1269, 1994.
42. C. Weibel. The K-book: an introduction to algebraic K-theory. Graduate Studies in Mathematics vol. 145, AMS, 2013