Enumeration and Updates for Conjunctive Linear Algebra Queries Through Expressibility

Thomas Muñoz Serrano
UHasselt, Data Science Institute, Belgium

Cristian Riveros
Pontificia Universidad Católica de Chile, Chile
Millennium Institute for Foundational Research on Data, Chile

Stijn Vansummeren
UHasselt, Data Science Institute, Belgium

Abstract

Due to the importance of linear algebra and matrix operations in data analytics, there is significant interest in using relational query optimization and processing techniques for evaluating (sparse) linear algebra programs. In particular, in recent years close connections have been established between linear algebra programs and relational algebra that allow transferring optimization techniques of the latter to the former. In this paper, we ask ourselves which linear algebra programs in MATLANG correspond to the free-connex and q-hierarchical fragments of conjunctive first-order logic. Both fragments have desirable query processing properties: free-connex conjunctive queries support constant-delay enumeration after a linear-time preprocessing phase, and q-hierarchical conjunctive queries further allow constant-time updates. By characterizing the corresponding fragments of MATLANG, we hence identify the fragments of linear algebra programs that one can evaluate with constant-delay enumeration after linear-time preprocessing and with constant-time updates. To derive our results, we improve and generalize previous correspondences between MATLANG and relational algebra evaluated over semiring-annotated relations. In addition, we identify properties on semirings that allow to generalize the complexity bounds for free-connex and q-hierarchical conjunctive queries from Boolean annotations to general semirings.

2012 ACM Subject Classification Theory of computation → Database theory

Keywords and phrases Query evaluation, conjunctive queries, linear algebra, enumeration algorithms.

Digital Object Identifier 10.4230/LIPIcs.ICDT.2024.12

Related Version *Full paper version with proofs*: `https://arxiv.org/abs/2310.04118` [38]

Acknowledgements Cristian Riveros was funded by ANID – Millennium Science Initiative Program – Code `ICN17_0` and ANID Fondecyt Regular project 1230935. Thomas Muñoz and Stijn Vansummeren were supported by the Bijzonder Onderzoeksfonds (BOF) of Hasselt University (Belgium) under Grants No. BOF21OWB13 and BOF20ZAP02 as well as by the Research Foundation Flanders (FWO) under research project Grant No. G019222N.

1 Introduction

Linear algebra forms the backbone of modern data analytics, as most machine learning algorithms are coded as sequences of matrix operations [1, 4, 19, 20, 42]. In practice, linear algebra programs operate over matrices with millions of entries. Therefore, efficient evaluation of linear algebra programs is a relevant challenge for data management systems which has attracted research attention with several proposals in the area [30, 33, 34, 37, 41].

To optimize and evaluate linear algebra programs, we must first agree on the language in which such programs are expressed. There has been a renewed interest in recent years for designing query languages for specifying linear algebra programs and for understanding their

expressive power [6,12,13,23,40]. One such proposal is MATLANG [12], a formal matrix query language that consists of only the basic linear algebra operations and whose extensions (e.g., for-MATLANG) achieve the expressive power of most linear algebra operations [23]. Although MATLANG is a theoretical query language, it includes the core of any linear algebra program and, thus, the optimization and efficient evaluation of MATLANG could have a crucial impact on today's machine learning systems.

In this work, we study the efficient evaluation of MATLANG programs over sparse matrices whose entries are taken from a general semiring. We consider MATLANG evaluation in both the static and dynamic setting. For static evaluation, we want to identify the fragment that one can evaluate by preprocessing the input in linear time to build a data structure for enumerating the output entries with constant-delay. For dynamic evaluation, we assume that matrix entries are updated regularly and we want to maintain the output of a MATLANG query without recomputing it. For this dynamic setting, we aim to identify the MATLANG fragment that one can evaluate by taking linear time in the size of the update to refresh the aforementioned data structure so that it supports constant-delay enumeration of the modified output entries. These guarantees for both scenarios have become the holy grail for algorithmic query processing since, arguably, it is the best that one can achieve complexity-wise in terms of the input, the output, and the cost of an update [5,8,11,16,17,31,35,36,39].

To identify the MATLANG fragments with these guarantees, our approach is straightforward but effective. Instead of developing evaluation algorithms from scratch, we establish a direct correspondence between linear algebra and relational algebra to take advantage of the query evaluation results for conjunctive queries. Indeed, prior work has precisely characterized which subfragments of conjunctive queries can be evaluated and updated efficiently [5,8,9,31]. Our main strategy, then, is to link these conjunctive query fragments to corresponding linear algebra fragments. More specifically, our contributions are as follows.

1. We start by understanding the deep connection between positive first-order logic (FO^+) over binary relations and sum-MATLANG [23], an extension of MATLANG. We formalize this connection by introducing schema encodings, which specify how relations simulate matrices and vice-versa, forcing a lossless relationship between both. Using this machinery, we show that sum-MATLANG and positive first-order logic are equally expressive over any relation, matrix, and matrix dimension (including non-rectangular matrices). Moreover, we show that conjunctive queries (CQ) coincide with sum-MATLANG without matrix addition, which we call conj-MATLANG. This result forms the basis for linking both settings and translating the algorithmic results from CQ to subfragments of conj-MATLANG.
2. We propose free-connex MATLANG (fc-MATLANG) for static evaluation, a natural MATLANG subfragment that we show to be equally expressive as *free-connex CQ* [5], a subfragment of CQ that allows linear time preprocessing and constant-delay enumeration. To obtain our expressiveness result, we show that free-connex CQs over binary relations are equally expressive as the two-variable fragment of conjunctive FO^+, a logical characterization of this class that could be of independent interest.
3. For the dynamic setting we introduce the language qh-MATLANG, a MATLANG fragment that we show equally expressive to *q-hierarchical CQ* [9,31], a fragment of CQ that allows constant update time and constant-delay enumeration.
4. Both free-connex and q-hierarchical CQ are known to characterize the class of CQs that one can evaluate efficiently on Boolean databases. We are interested, however, in evaluating MATLANG queries on matrices featuring entries in a *general semiring*. To obtain the complexity bounds for fc-MATLANG and qh-MATLANG on general semirings, therefore, we show that the upper and lower bounds for free-connex and q-hierarchical

CQs generalize from Boolean annotations to classes of semirings which includes most semirings used in practice, like the reals. The tight expressiveness connections established in this paper then prove that for such semirings fc-MATLANG and qh-MATLANG can be evaluated with the same guarantees as their CQ counterparts and that they are optimal: one cannot evaluate any other conj-MATLANG query outside this fragment under complexity-theoretic assumptions [9].

An extended version of this paper that includes full proofs of formal statements is available online [38].

Related work. In addition to the work that has already been cited above, the following work is relevant. Brijder et al. [13] have shown equivalence between MATLANG and FO_3^+, the 3-variable fragment of positive first order logic. By contrast, we show equivalence between sum-MATLANG and FO^+, and study the relationship between the free-connex and q-hierarchical fragments of MATLANG and $\mathsf{FO}^\wedge$, the conjunctive fragment of positive first order logic.

Geerts et al. [23] previously established a correspondence between sum-MATLANG and FO^+. However, as we illustrate in [38] their correspondence is (1) restricted to square matrices, (2) asymmetric between the two settings, and (3) encodes matrix instances as databases of more than linear size, making it unsuitable to derive the complexity bounds.

Eldar et al. [18] have recently also generalized complexity bounds for free-connex CQs from Boolean annotations to general semirings. Nevertheless, this generalization is with respect to *direct access*, not enumeration. In their work the focus is to compute aggregate queries, which is achieved by providing direct access to the answers of a query even if the annotated value (aggregation result) is zero. By contrast, in our setting, zero-annotated values must not be reported during the enumeration of query answers. This difference in the treatment of zero leads to a substantial difference in the properties that a semiring must have in order to generalize the existing complexity bounds.

There are deep connections known between the treewidth and the number of variables of a conjunctive FO^+ formula ($\mathsf{FO}^\wedge$). For example, Kolaitis and Vardi established the equivalence of boolean queries in $\mathsf{FO}_k^\wedge$, the k-variable fragment of $\mathsf{FO}^\wedge$, and boolean queries in $\mathsf{FO}^\wedge$ of treewidth less than k. Because they focus on boolean queries (i.e., without free variables), this result does not imply our result that for binary queries free-connex $\mathsf{FO}^\wedge$ equals $\mathsf{FO}_2^\wedge$. Similarly, Geerts and Reutter [24] introduce a tensor logic TL over binary relations and show that conjunctive expressions in this language that have treewidth k can be expressed in TL_{k+1}, the k-variable fragment of TL. While they do take free variables into account, we show in [38] that there are free-connex conjunctive queries with 2 free variables with treewidth 2 in their formalism—for which their result hence only implies expressibility in $\mathsf{FO}_3^\wedge$, not $\mathsf{FO}_2^\wedge$ as we show here.

Several proposals [30, 33, 34, 37, 41] have been made regarding the efficient evaluation of linear algebra programs in the last few years. All these works focused on query optimization without formal guarantees regarding the preprocessing, updates, or enumeration in query evaluation. To the best of our knowledge, this is the first work on finding subfragments of a linear algebra query language (i.e., MATLANG) with such efficient guarantees.

2 Preliminaries

In this section we recall the main definitions of MATLANG, a query language on matrices, and first order logic (FO), a query language on relations.

Semirings. We evaluate both languages over arbitrary commutative and non-trivial semirings. A (commutative and non-trivial) *semiring* $(K, \oplus, \odot, \mathbb{0}, \mathbb{1})$ is an algebraic structure where K is a non-empty set, $\oplus$ and $\odot$ are binary operations over K, and $\mathbb{0}, \mathbb{1} \in K$ with $\mathbb{0} \neq \mathbb{1}$. Furthermore, $\oplus$ and $\odot$ are associative operations, $\mathbb{0}$ and $\mathbb{1}$ are the identities of $\oplus$ and $\odot$, respectively, $\oplus$ and $\odot$ are commutative operations, $\odot$ distributes over $\oplus$, and $\mathbb{0}$ annihilates K (i.e. $\mathbb{0} \odot k = k \odot \mathbb{0} = \mathbb{0}$). We use $\bigoplus_L$ and $\bigodot_L$ to denote the $\oplus$ and $\odot$ operation over all elements in $L \subseteq K$, respectively. Typical examples of semirings are the reals $(\mathbb{R}, +, \times, 0, 1)$, the natural numbers $(\mathbb{N}, +, \times, 0, 1)$, and the boolean semiring $\mathbb{B} = (\{\mathtt{t}, \mathtt{f}\}, \vee, \wedge, \mathtt{f}, \mathtt{t})$.

Henceforth, when we say "semiring" we mean "commutative and non-trivial" semiring. We fix such an arbitrary semiring K throughout the document. We denote by $\mathbb{N}_{>0}$ the set of non-zero natural numbers.

Matrices and size symbols. A *K-matrix* (or just matrix) of dimension $m \times n$ is a $m \times n$ matrix with elements in K as its entries. We write $\mathbf{A}_{ij}$ to denote the (i, j)-entry of $\mathbf{A}$. Matrices of dimension $m \times 1$ are *column vectors* and those of dimension $1 \times n$ are *row vectors*. We also refer to matrices of dimension 1×1 as *scalars*.

We assume a collection of *size symbols* denoted with greek letters $\alpha, \beta, \ldots$ and assume that the natural number 1 is a valid size symbol. A *type* is a pair (α, β) of size symbols. Intuitively, types represent sets of matrix dimensions. In particular, we obtain dimensions from types by replacing size symbols by elements from $\mathbb{N}_{>0}$, where the size symbol 1 is always replaced by the natural number 1. So, (α, β) with $\alpha \neq 1 \neq \beta$ represents the set of dimensions $\{(m, n) \mid m, n \in \mathbb{N}_{>0}\}$, while (α, α) represents the dimensions $\{(m, m) \mid m \in \mathbb{N}_{>0}\}$ of square matrices; and $(\alpha, 1)$ represents the dimensions $\{(m, 1) \mid m \in \mathbb{N}_{>0}\}$ of column vectors and $(1, 1)$ represents the dimension $(1, 1)$ of scalars.

Schemas and instances. We assume a set $\mathcal{M} = \{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{V}, \ldots\}$ of *matrix symbols*, disjoint with the size symbols and denoted by bold uppercase letters. Each matrix symbol $\mathbf{A}$ has a fixed associated type. We write $\mathbf{A}\colon (\alpha, \beta)$ to denote that $\mathbf{A}$ has type (α, β).

A matrix *schema* $\mathcal{S}$ is a finite set of matrix and size symbols. We require that the special size symbol 1 is always in $\mathcal{S}$, and that all size symbols occurring in the type of any matrix symbol $\mathbf{A} \in \mathcal{S}$ are also in $\mathcal{S}$. A matrix *instance* $\mathcal{I}$ over a matrix schema $\mathcal{S}$ is a function that maps each size symbol α in $\mathcal{S}$ to a non-zero natural number $\alpha^{\mathcal{I}} \in \mathbb{N}_{>0}$, and maps each matrix symbol $\mathbf{A}\colon (\alpha, \beta)$ in $\mathcal{S}$ to a K-matrix $\mathbf{A}^{\mathcal{I}}$ of dimension $\alpha^{\mathcal{I}} \times \beta^{\mathcal{I}}$. We assume that for the size symbol 1, we have $1^{\mathcal{I}} = 1$, for every instance $\mathcal{I}$.

Sum-Matlang. Let $\mathcal{S}$ be a matrix schema. Before defining the syntax of sum-MATLANG, we assume a set $\mathcal{V} = \{\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{x}, \ldots\}$ of *vector variables* over $\mathcal{S}$, which is disjoint with matrix and size symbols in $\mathcal{S}$. Each such variable $\mathbf{v}$ has a fixed associated type, which must be a vector type $(\gamma, 1)$ for some size symbol $\gamma \in \mathcal{S}$. We also write $\mathbf{v}\colon (\gamma, 1)$ in that case.

The syntax of sum-MATLANG expressions [23] over $\mathcal{S}$ is defined by the following grammar:

$$
\begin{array}{rcllll}
e & ::= & \mathbf{A} \in \mathcal{S} & \text{(matrix symbol)} & \mid \ \mathbf{v} \in \mathcal{V} & \text{(vector variable)} \\
 & \mid & e^T & \text{(transpose)} & \mid \ e_1 \cdot e_2 & \text{(matrix multiplication)} \\
 & \mid & e_1 + e_2 & \text{(matrix addition)} & \mid \ e_1 \times e_2 & \text{(scalar multiplication)} \\
 & \mid & e_1 \odot e_2 & \text{(pointwise multiplication)} & \mid \ \Sigma \mathbf{v}.e & \text{(sum-iteration).}
\end{array}
$$

In addition, we require that expressions e are *well-typed*, in the sense that its *type* $\mathtt{type}(e)$ is

correctly defined as follows:

$$
\begin{aligned}
\texttt{type}(\mathbf{A}) &\coloneqq (\alpha,\beta) && \text{for a matrix symbol } \mathbf{A}\colon (\alpha,\beta)\\
\texttt{type}(\mathbf{v}) &\coloneqq (\gamma,1) && \text{for vector variable } \mathbf{v}\colon (\gamma,1)\\
\texttt{type}(e^T) &\coloneqq (\beta,\alpha) && \text{if } \texttt{type}(e)=(\alpha,\beta)\\
\texttt{type}(e_1\cdot e_2) &\coloneqq (\alpha,\gamma) && \text{if } \texttt{type}(e_1)=(\alpha,\beta) \text{ and } \texttt{type}(e_2)=(\beta,\gamma)\\
\texttt{type}(e_1+e_2) &\coloneqq (\alpha,\beta) && \text{if } \texttt{type}(e_1)=\texttt{type}(e_2)=(\alpha,\beta)\\
\texttt{type}(e_1\times e_2) &\coloneqq (\alpha,\beta) && \text{if } \texttt{type}(e_1)=(1,1) \text{ and } \texttt{type}(e_2)=(\alpha,\beta)\\
\texttt{type}(e_1\odot e_2) &\coloneqq (\alpha,\beta) && \text{if } \texttt{type}(e_1)=\texttt{type}(e_2)=(\alpha,\beta)\\
\texttt{type}(\Sigma\mathbf{v}.e) &\coloneqq (\alpha,\beta) && \text{if } e\colon (\alpha,\beta) \text{ and } \texttt{type}(\mathbf{v})=(\gamma,1).
\end{aligned}
$$

In what follows, we always consider well-typed expressions and write $e\colon(\alpha,\beta)$ to denote that e is well typed, and its type is (α,β).

For an expression e, we say that a vector variable $\mathbf{v}$ is *bound* if it is under a sum-iteration $\Sigma\mathbf{v}$, and *free* otherwise. To evaluate expressions with free vector variables, we require the following notion of a valuation. Fix a matrix instance $\mathcal{I}$ over $\mathcal{S}$. A *vector valuation* over $\mathcal{I}$ is a function μ that maps each vector symbol $\mathbf{v}\colon(\gamma,1)$ to a column vector of dimension $\gamma^{\mathcal{I}}\times 1$. Further, if $\mathbf{b}$ is a vector of dimension $\gamma^{\mathcal{I}}\times 1$, then let $\mu[\mathbf{v}:=\mathbf{b}]$ denote the *extended* vector valuation over $\mathcal{I}$ that coincides with μ, except that $\mathbf{v}\colon(\gamma,1)$ is mapped to $\mathbf{b}$.

Let $e\colon(\alpha,\beta)$ be a sum-MATLANG expression over $\mathcal{S}$. When one evaluates e over a matrix instance $\mathcal{I}$ and a matrix valuation μ over $\mathcal{I}$, it produces a matrix $[\![e]\!](\mathcal{I},\mu)$ of dimension $\alpha^{\mathcal{I}}\times\beta^{\mathcal{I}}$ such that each entry i,j satisfies:

$$
\begin{aligned}
[\![\mathbf{A}]\!](\mathcal{I},\mu)_{ij} &\coloneqq \mathbf{A}^{\mathcal{I}}_{ij} \text{ for } \mathbf{A}\in\mathcal{S}\\
[\![\mathbf{v}]\!](\mathcal{I},\mu)_{ij} &\coloneqq \mu(\mathbf{v})_{ij} \text{ for } \mathbf{v}\in\mathcal{V}\\
[\![e^T]\!](\mathcal{I},\mu)_{ij} &\coloneqq [\![e]\!](\mathcal{I},\mu)_{ji}\\
[\![e_1+e_2]\!](\mathcal{I},\mu)_{ij} &\coloneqq [\![e_1]\!](\mathcal{I},\mu)_{ij}\oplus[\![e_2]\!](\mathcal{I},\mu)_{ij}\\
[\![e_1\odot e_2]\!](\mathcal{I},\mu)_{ij} &\coloneqq [\![e_1]\!](\mathcal{I},\mu)_{ij}\odot[\![e_2]\!](\mathcal{I},\mu)_{ij}\\
[\![\Sigma\mathbf{v}.\,e\,]\!](\mathcal{I},\mu)_{ij} &\coloneqq \textstyle\bigoplus_{k=1}^{\gamma^{\mathcal{I}}}[\![e]\!](\mathcal{I},\mu[\mathbf{v}:=\mathbf{b}_k^{\gamma^{\mathcal{I}}}])_{ij}\\
[\![e_1\cdot e_2]\!](\mathcal{I},\mu)_{ij} &\coloneqq \textstyle\bigoplus_{k}[\![e_1]\!](\mathcal{I},\mu)_{ik}\odot[\![e_2]\!](\mathcal{I},\mu)_{kj}\\
[\![e_1\times e_2]\!](\mathcal{I},\mu)_{ij} &\coloneqq a\odot[\![e_2]\!](\mathcal{I},\mu)_{ij} \text{ with } [\![e_1]\!](\mathcal{I},\mu)=[a]
\end{aligned}
$$

where $\mathbf{v}\colon(\gamma,1)$ and $\mathbf{b}^n_1,\mathbf{b}^n_2,\ldots,\mathbf{b}^n_n$ are the n-dimension canonical vectors, namely, the vectors $[\mathbb{1}\,\mathbb{0}\,\ldots\,\mathbb{0}]^T$, $[\mathbb{0}\,\mathbb{1}\,\ldots\,\mathbb{0}]^T$, …, $[\mathbb{0}\,\mathbb{0}\,\ldots\,\mathbb{1}]^T$, respectively.

▶ **Example 1.** Let $\mathcal{S}=\{\mathbf{A}\}$ where $\mathbf{A}\colon(\alpha,\alpha)$. Let $\mathcal{I}$ be an instance over $\mathcal{S}$ such that $\alpha^{\mathcal{I}}=3$ and $\mathbf{A}^{\mathcal{I}}=\begin{bmatrix}a_{11}&a_{12}&a_{13}\\a_{21}&a_{22}&a_{23}\\a_{31}&a_{32}&a_{33}\end{bmatrix}$. Let $\mathbf{v}\in\mathcal{V}$ where $\mathbf{v}\colon(\alpha,1)$. The expression $\Sigma\mathbf{v}.\mathbf{A}\cdot\mathbf{v}$ is well-typed and

$$
[\![\Sigma\mathbf{v}.\mathbf{A}\cdot\mathbf{v}]\!](\mathcal{I},\emptyset)=\mathbf{A}\cdot\mathbf{b}^3_1+\mathbf{A}\cdot\mathbf{b}^3_2+\mathbf{A}\cdot\mathbf{b}^3_3=\begin{bmatrix}a_{11}\\a_{21}\\a_{31}\end{bmatrix}+\begin{bmatrix}a_{12}\\a_{22}\\a_{32}\end{bmatrix}+\begin{bmatrix}a_{13}\\a_{23}\\a_{33}\end{bmatrix}.
$$

▶ **Example 2.** Let $\mathcal{S}$ and $\mathcal{I}$ be as in Example 1. Let $\mathbf{v}\in\mathcal{V}$ where $\mathbf{v}\colon(\gamma,1)$. The expression $\Sigma\mathbf{v}.\mathbf{A}$ is well-typed and

$$
[\![\Sigma\mathbf{v}.\mathbf{A}]\!](\mathcal{I},\emptyset)=\underbrace{\mathbf{A}+\cdots+\mathbf{A}}_{\gamma^{\mathcal{I}}\text{ times}}.
$$

Matlang. MATLANG is a linear algebra query language that is a fragment of sum-MATLANG. Specifically, define the syntax of MATLANG expressions over $\mathcal{S}$ by the following grammar:

$$e \quad ::= \quad \mathbf{A} \in \mathcal{S} \mid e^T \mid e_1 \cdot e_2 \mid e_1 + e_2 \mid e_1 \times e_2 \mid e_1 \odot e_2 \mid \mathbf{1}^\alpha \mid \mathbf{I}^\alpha$$

for every size symbol $\alpha \in \mathcal{S}$. Here, $\mathbf{1}^\alpha$ and $\mathbf{I}^\alpha$ denote the *ones-vector* and *identity-matrix*, respectively, of type $\mathtt{type}(\mathbf{1}^\alpha) = (\alpha, 1)$ and $\mathtt{type}(\mathbf{I}^\alpha) = (\alpha, \alpha)$. Their semantics can be defined by using sum-MATLANG as:

$$[\![\mathbf{1}^\alpha]\!](\mathcal{I}, \mu) \coloneqq [\![\Sigma \mathbf{v}.\mathbf{v}]\!](\mathcal{I}, \mu) \qquad [\![\mathbf{I}^\alpha]\!](\mathcal{I}, \mu) \coloneqq [\![\Sigma \mathbf{v}.\mathbf{v} \cdot \mathbf{v}^T]\!](\mathcal{I}, \mu)$$

Note that the original MATLANG version introduced in [12] included an operator for the *diagonalization of a vector*. This operator can be simulated by using the $\mathbf{I}^\alpha$-operator and vice versa. Furthermore, we have included the pointwise multiplication $\odot$ in MATLANG, also known as the *Hadamard product*. This operation will be essential for our characterization results. In [12, 23], the syntax of MATLANG was more generally parameterized by a family of n-ary functions that could be pointwise applied. Similarly to [13, 22] we do not include such functions here, but leave their detailed study to future work.

Sum-Matlang queries. A sum-MATLANG *query* $\mathbf{Q}$ over a matrix schema $\mathcal{S}$ is an expression of the form $\mathbf{H} \coloneqq e$ where e is a well-typed sum-MATLANG expression without free vector variables, $\mathbf{H}$ is a "fresh" matrix symbol that does not occur in $\mathcal{S}$, and $\mathtt{type}(e) = \mathtt{type}(\mathbf{H})$. When evaluated on a matrix instance $\mathcal{I}$ over schema $\mathcal{S}$, $\mathbf{Q}$ returns a matrix instance $\mathcal{E}$ over the extended schema $\mathcal{S} \cup \{\mathbf{H}\}$: $\mathcal{E}$ coincides with $\mathcal{I}$ for every matrix and size symbol in $\mathcal{S}$ and additionally maps $\mathbf{H}^{\mathcal{E}} = [\![e]\!](\mathcal{I}, \emptyset)$ with $\emptyset$ denoting the empty vector valuation. We denote the instance resulting from evaluating $\mathbf{Q}$ by $[\![\mathbf{Q}]\!](\mathcal{I})$. If $\mathcal{S}$ is a matrix schema and $\mathbf{Q}$ a sum-MATLANG query over $\mathcal{S}$ then we use $\mathcal{S}(\mathbf{Q})$ to denote the extended schema $\mathcal{S} \cup \{\mathbf{H}\}$.

K-relations. A *K-relation* over a domain of data values $\mathbb{D}$ is a function $f \colon \mathbb{D}^a \to K$ such that $f(\overline{d}) \neq \mathbb{0}$ for finitely many $\overline{d} \in \mathbb{D}^a$. Here, '$a$' is the *arity* of R. Since we want to compare relational queries with sum-MATLANG queries, we will restrict our attention in what follows to K-relations where the domain $\mathbb{D}$ of data values is the set $\mathbb{N}_{>0}$. In this context, we may naturally view a K-matrix of dimensions $n \times m$ as a K-relation such that the entry (i, j) of the matrix is encoded by the K-value of the tuple (i, j) in the relation (see also Section 3).

Vocabularies and databases. We assume an infinite set of *relation symbols* together with an infinite and disjoint set of *constant symbols*. Every relation symbol R is associated with a number, its *arity*, which we denote by $ar(R) \in \mathbb{N}$. A *vocabulary* σ is a finite set of relation and constant symbols. A *database* over σ is a function db that maps every constant symbol $c \in \sigma$ to a value c^{db} in $\mathbb{N}_{>0}$; and every relation symbol $R \in \sigma$ to a K-relation R^{db} of arity $ar(R)$.

Positive first order logic. As our relational query language, we will work with the positive fragment of first order logic (FO^+). In contrast to the standard setting in database theory, where the only atomic formulas are relational atoms of the form $R(\overline{x})$, we also allow the ability to compare variables with constant symbols. To this end, the following definitions are in order. We assume an infinite set of variables, which we usually denote by x, y, z. We denote tuples of variables by $\overline{x}$, $\overline{y}$, and so on. A *relational atom* is expression of the form $R(x_1, \ldots, x_k)$ with R a relation symbol of arity k. A *comparison atom* is of the form $x \leq c$ with x a variable and c a constant symbol. A *positive first order logic formula* (FO^+ formula) over a vocabulary σ is an expression generated by the following grammar:

$$\varphi \quad ::= \quad R(\overline{x}) \mid x \leq c \mid \exists \overline{y}.\, \varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$$

where R and c range over relations and constants in σ, respectively. We restrict ourselves to *safe* formulas, in the sense that in a disjunction $\varphi_1 \vee \varphi_2$, we require that φ_1 and φ_2 have the same set of free variables [2]. The notions of *free and bound variables* are defined as usual in FO. We denote the set of free variables of φ by $\mathit{free}(\varphi)$ and its multiset of atoms by $\mathit{at}(\varphi)$.

We evaluate formulas over K-relations as follows using the well-known semiring semantics [26]. A *valuation* over a set of variables X is a function $\nu \colon X \to \mathbb{N}_{>0}$ that assigns a value in $\mathbb{N}_{>0}$ to each variable in X (recall that $\mathbb{D} = \mathbb{N}_{>0}$). We denote by $\nu \colon X$ that ν is a valuation on X and by $\nu|_Y$ the restriction of ν to $X \cap Y$. As usual, valuations are extended point-wise to tuples of variables, i.e., $\nu(x_1, \ldots, x_n) = (\nu(x_1), \ldots, \nu(x_n))$. Let φ be an FO^+ formula over vocabulary σ. When evaluated on a database db over vocabulary σ, it defines a mapping $[\![\varphi]\!]_{db}$ from valuations over $\mathit{free}(\varphi)$ to K inductively defined as:

$$\begin{aligned}
[\![R(x_1, \ldots, x_k)]\!]_{db}(\nu) &\coloneqq R^{db}(\nu(x_1), \ldots, \nu(x_n)) \\
[\![x \leq c]\!]_{db}(\nu) &\coloneqq \begin{cases} \mathbb{1} & \text{if } \nu(x) \leq c^{db} \\ \mathbb{0} & \text{otherwise} \end{cases} \\
[\![\varphi_1 \wedge \varphi_2]\!]_{db}(\nu) &\coloneqq [\![\varphi_1]\!]_{db}(\nu|_{\mathit{free}(\varphi_1)}) \odot [\![\varphi_2]\!]_{db}(\nu|_{\mathit{free}(\varphi_2)}) \\
[\![\varphi_1 \vee \varphi_2]\!]_{db}(\nu) &\coloneqq [\![\varphi_1]\!]_{db}(\nu) \oplus [\![\varphi_2]\!]_{db}(\nu) \\
[\![\exists y.\varphi]\!]_{db}(\nu) &\coloneqq \bigoplus\nolimits_{\mu \colon \mathit{free}(\varphi) \text{s.t.} \mu|_{\mathit{free}(\varphi)\setminus\{y\}} = \nu} [\![\varphi]\!]_{db}(\mu)
\end{aligned}$$

FO queries. An FO^+ *query* Q over vocabulary σ is an expression of the form $H(\overline{x}) \leftarrow \varphi$ where φ is an FO^+ formula over σ, $\overline{x} = (x_1, \ldots, x_k)$ is a sequence of (not necessarily distinct) free variables of φ, such that every free variable of φ occurs in $\overline{x}$, and H is a "fresh" relation symbol not in σ with $\mathit{ar}(H) = k$. The formula φ is called the *body* of Q, and $H(\overline{x})$ its *head*.

When evaluated over a database db over σ, Q returns a database $[\![Q]\!]_{db}$ over the extended vocabulary $\sigma \cup \{H\}$. This database $[\![Q]\!]_{db}$ coincides with db for every relation and constant symbol in σ, and maps the relation symbol H to the K-relation of arity k defined as follows. For a sequence of domain values $\overline{d} = (d_1, \ldots, d_k)$, we write $\overline{d} \models \overline{x}$ if, for all $i \neq j$ with $x_i = x_j$ we also have $d_i = d_j$. Clearly, if $\overline{d} \models \overline{x}$ then the mapping $\{x_1 \to d_1, \ldots, x_k \mapsto d_k\}$ is well-defined. Denote this mapping by $\overline{x} \mapsto \overline{d}$ in this case. Then

$$[\![Q]\!]_{db}(H) \coloneqq \overline{d} \mapsto \begin{cases} [\![\varphi]\!]_{db}(\overline{x} \mapsto \overline{d}) & \text{if } \overline{d} \models \overline{x} \\ \mathbb{0} & \text{otherwise} \end{cases}$$

In what follows, if Q is a query, then we will often use the notation $Q(\overline{x})$ to denote that the sequence of the variables in the head of Q is $\overline{x}$. If Q is a query over σ and $H(\overline{x})$ its head, then we write $\sigma(Q)$ for the extended vocabulary $\sigma \cup \{H\}$.

We denote by $\mathsf{FO}^\wedge$ the fragment of FO^+ formulas in which disjunction is disallowed. A query $Q = H(\overline{x}) \leftarrow \varphi$ is an $\mathsf{FO}^\wedge$ query if φ is in $\mathsf{FO}^\wedge$. If additionally φ is in prenex normal form, i.e., $Q : H(\overline{x}) \leftarrow \exists \overline{y}.a_1 \wedge \cdots \wedge a_n$ with $a_1, \ldots, a_n$ (relational or comparison) atoms, then Q is a *conjunctive query* (CQ). Note that, while the classes of conjunctive queries and $\mathsf{FO}^\wedge$ queries are equally expressive, for our purposes conjunctive queries are hence formally a syntactic fragment of $\mathsf{FO}^\wedge$ queries.

An FO^+ query is *binary* if every relational atom occurring in it (body and head) has arity at most two. Because in sum-MATLANG both the input and output are matrices, our correspondences between sum-MATLANG and FO^+ will focus on binary queries.

Discussion. We have added comparison atoms to FO^+ in order to establish its correspondence with sum-MATLANG. To illustrate why we will need comparison atoms, consider the sum-MATLANG expression $\mathbf{I}^\alpha$ of type (α, α) that computes the identity matrix. This can be expressed by means of the following CQ $Q : I(x, x) \leftarrow x \leq \alpha$. We hence use comparison atoms to align the dimension of the matrices with the domain size of relations.

To make the correspondence hold, we note that in MATLANG there is a special size symbol, 1, which is always interpreted as the constant $1 \in \mathbb{N}$. This size symbol is used in particular to represent column and row vectors, which have type $(\alpha, 1)$ and $(1, \alpha)$ respectively. We endow CQs with the same property in the sense that we will assume in what follows that 1 is a valid constant symbol and that $1^{db} = 1$ for every database db.

3 From matrices to relations and back

Geerts et al. [23] previously established a correspondence between sum-MATLANG and FO^+. However, as we illustrate in the full version [38] their correspondence is (1) for square matrices, (2) asymmetric, and (3) encodes matrix instances as databases of more than linear size, making it unsuitable to derive the complexity bounds that we are interested in here. In this section, we revisit and generalize the connection between sum-MATLANG and FO^+ by providing translations between the two query languages that works for any matrix schema, are symmetric, and ensure that matrices are encoded as databases of linear size. Towards this goal, we introduce next all the formal machinery to link both settings. We start by determining precisely in what sense relations can encode matrices, or matrices can represent relations, and how this correspondence transfers to queries. Then we show how to generalize the expressibility results in [23] for any matrix sizes and every encoding between schemas.

How we relate objects. Let $\mathbf{A}$ be a matrix of dimension $m \times n$. There exist multiple natural ways to encode $\mathbf{A}$ as a relation, depending on the dimension $m \times n$.

- We can always encode $\mathbf{A}$, whatever the values of m and n, as the binary K-relation R such that (1) $\mathbf{A}_{i,j} = R(i, j)$ for every $i \leq m$, $j \leq n$ and (2) $R(i, j) = \mathbb{0}$ if $i > m$ or $j > n$.
- If $\mathbf{A}$ is a column vector ($n = 1$) then we can also encode it as the unary K-relation R such that $\mathbf{A}_{i,1} = R(i)$ for every $i \leq m$ and $R(i) = \mathbb{0}$ if $i > m$.
- Similarly, if $\mathbf{A}$ is a row vector ($m = 1$) then we can encode it as the unary K-relation R with $\mathbf{A}_{1,j} = R(j)$ for every $j \leq n$ and $R(i) = \mathbb{0}$ if $j > n$.
- If $\mathbf{A}$ is a scalar ($m = n = 1$), we can encode it as a nullary K-relation R with $\mathbf{A}_{1,1} = R()$.

Note that if $\mathbf{A}$ is scalar then we can hence encode it by means of a binary relation, a unary relation, or a nullary relation; and if it is a vector we can encode it by a binary or unary relation. In what follows, we write $\mathbf{A} \simeq R$ to denote that R encodes $\mathbf{A}$.

Conversely, given a (nullary, unary, or binary) K-relation R we may interpret this as a matrix of appropriate dimension. Specifically, we say that relation R is *consistent* with dimension $m \times n$ if there exists a matrix $\mathbf{A}$ of dimension $m \times n$ such that $\mathbf{A} \simeq R$. This is equivalent to requiring that relation is $\mathbb{0}$ on entries outside of $m \times n$. Note that, given R that is consistent with $m \times n$ there is exactly one matrix $\mathbf{A} \colon m \times n$ such that $\mathbf{A} \simeq R$.

How we relate schemas. A *matrix-to-relational schema encoding* from a matrix schema $\mathcal{S}$ to a relational vocabulary σ is a function $Rel \colon \mathcal{S} \to \sigma$ that maps every matrix symbol $\mathbf{A}$ in $\mathcal{S}$ to a unary or binary relation symbol $Rel(\mathbf{A})$ in σ, and every size symbol α in $\mathcal{S}$ to a constant symbol $Rel(\alpha)$ in σ. Here, $Rel(\mathbf{A})$ can be unary only if $\mathbf{A}$ is of vector type, and nullary only if $\mathbf{A}$ is of scalar type. Intuitively, Rel specifies which relation symbols will be used to store

the encodings of which matrix symbols. In addition, we require that $Rel(1) = 1$ and that Rel is a bijection between $\mathcal{S}$ and σ. This makes sure that we can always invert Rel. In what follows, we will only specify that Rel is a matrix-to-relational schema encoding *on* $\mathcal{S}$, leaving the vocabulary σ unspecified. In that case, we write $Rel(\mathcal{S})$ for the relational vocabulary σ.

Conversely, we define a *relational-to-matrix schema encoding* from σ into $\mathcal{S}$ as a function $Mat\colon \sigma \to \mathcal{S}$ that maps every relation symbol R to a matrix symbol $Mat(R)$ and every constant symbol c to a size symbol $Mat(c)$. We require that all unary relations are mapped to matrix symbols of vector type, either row or column, and that all nullary relations are mapped to matrix symbols of scalar type. Furthermore, Mat must map $1 \mapsto 1$ and be bijective. Similarly, we denote by $Mat(\sigma)$ the matrix schema $\mathcal{S}$ mapped by Mat.

Note that the bijection assumption over Mat imposes some requirements over σ to encode it as matrices. For example, Mat requires the existence of at least one constant symbol in σ for encoding matrices dimensions, since every matrix symbol has at least one size symbol in its type, and that size symbol is by definition in $\mathcal{S}$. The bijection between constant and size symbols is necessary in order to have lossless encoding between both settings.

Given that Rel and Mat are bijections between $\mathcal{S}$ and σ, their inverses Rel^{-1} and Mat^{-1} are well defined. Furthermore, by definition we have that Rel^{-1} and Mat^{-1} are relational-to-matrix and matrix-to-relational schema encodings, respectively.

How we relate instances. We start by specifying how to encode matrix instances as database instances. Fix a matrix-to-relational schema encoding Rel between $\mathcal{S}$ and $Rel(\mathcal{S})$. Let $\mathcal{I}$ be a matrix instance over $\mathcal{S}$ and db a database over $Rel(\mathcal{S})$. We say that db *is a relational encoding of* $\mathcal{I}$ *w.r.t.* Rel, denoted by $\mathcal{I} \simeq_{Rel} db$, if

- $\mathbf{A}^{\mathcal{I}} \simeq Rel(\mathbf{A})^{db}$ for every matrix symbol $\mathbf{A}$ in $\mathcal{S}$, and
- $Rel(\alpha)^{db} = \alpha^{\mathcal{I}}$ for every size symbol α in $\mathcal{S}$.

Note that, given $\mathcal{I}$ and Rel, the relational encoding db is uniquely defined. As such, we also denote this database by $Rel(\mathcal{I})$.

We now focus on interpreting database instances as matrix instances, which is more subtle. Fix a relational-to-matrix schema encoding Mat from σ to $Mat(\sigma)$. We need to first leverage the consistency requirement from relations to databases. Formally, we say that a database db over σ is *consistent with* Mat if for every relation symbol R in σ, R^{db} is consistent with dimension $c^{db} \times d^{db}$ where $Mat(R)\colon (Mat(c), Mat(d))$. In other words, a consistent database specifies the value of each dimension, and the relations are themselves consistent with them.

Let db be a database over σ, consistent with Mat and let $\mathcal{I}$ be a matrix instance of $Mat(\sigma)$. We say that $\mathcal{I}$ *is a matrix encoding of* db *w.r.t.* Mat, denoted $db \simeq_{Mat} \mathcal{I}$, if

- $Mat(R)^{\mathcal{I}} \simeq R^{db}$ for every relation symbol $R \in \sigma$; and
- $c^{db} = Mat(c)^{\mathcal{I}}$ for every constant symbol $c \in \sigma$.

Given Mat and a consistent database db, the matrix encoding $\mathcal{I}$ is uniquely defined. As such, we also denote this instance by $Mat(db)$.

From the previous definitions, one notes an asymmetry between both directions. Although an encoding always holds from matrices to relations, we require that the relations are consistent with the sizes (i.e., constants) from relations to matrices. Nevertheless, this asymmetry does not impose a problem when we want to go back and forth, as the next result shows.

▶ **Proposition 3.** *Let Rel and Mat be matrix-to-relational and relational-to-matrix schema encodings from $\mathcal{S}$ to σ and from σ to $\mathcal{S}$, respectively, such that $Mat = Rel^{-1}$. Then*

- *$Rel^{-1}(Rel(\mathcal{S})) = \mathcal{S}$ and $Mat^{-1}(Mat(\sigma)) = \sigma$;*
- *$Rel(\mathcal{I})$ is consistent with Rel^{-1}, for every instance $\mathcal{I}$ over $\mathcal{S}$;*
- *$Rel^{-1}(Rel(\mathcal{I})) = \mathcal{I}$, for every instance $\mathcal{I}$ over $\mathcal{S}$; and*

- $Mat^{-1}(Mat(db)) = db$, *for every db consistent with Mat.*

The previous proposition is a direct consequence of the definitions; however, it shows that the consistency requirement and schema encodings provide a lossless encoding between the relational and matrix settings. This fact is crucial to formalize the expressiveness equivalence between sum-MATLANG and FO^+, and their subfragments in the following sections.

From sum-Matlang to positive-FO. We first aim to simulate every sum-MATLANG query with an FO^+ query w.r.t. some matrix-to-relational schema encoding. This was already proven in [23] for a different but related setting, and only for *Rel* encodings that map matrix symbols with vector types into unary relations. Here we generalize it to arbitrary encodings.

In what follows, fix a matrix schema $\mathcal{S}$. Let $\mathbf{Q}$ be a sum-MATLANG query over $\mathcal{S}$, and *Rel* be a matrix-to-relational schema encoding on $\mathcal{S}(\mathbf{Q})$. We say that FO^+ query Q *simulates* $\mathbf{Q}$ w.r.t. *Rel* if $Rel(\llbracket \mathbf{Q} \rrbracket(\mathcal{I})) = \llbracket Q \rrbracket_{Rel(\mathcal{I})}$ for every matrix instance $\mathcal{I}$ over $\mathcal{S}$. Note that the definition implies that the output matrix symbol of $\mathbf{Q}$ must be mapped to the output relation symbol of Q by *Rel*, since *Rel* is a bijection and the condition must hold for every matrix instance. Indeed, it is equivalent to $\llbracket \mathbf{Q} \rrbracket(\mathcal{I}) = Rel^{-1}(\llbracket Q \rrbracket_{Rel(\mathcal{I})})$, namely, that one can evaluate $\mathbf{Q}$ by first evaluating $\llbracket Q \rrbracket_{Rel(\mathcal{I})}$ and then mapping the results back.

Next, we show that we can simulate every sum-MATLANG query in the relational setting.

▶ **Proposition 4.** *For every* sum-MATLANG *query* $\mathbf{Q}$ *over* $\mathcal{S}$ *and every matrix-to-relational schema encoding Rel on* $\mathcal{S}(\mathbf{Q})$*, there exists an* FO^+ *query* Q *that simulates* $\mathbf{Q}$ *w.r.t. Rel.*

From positive-FO to sum-Matlang. We now aim to simulate every FO^+ query with a sum-MATLANG query. Contrary to the previous direction, the expressiveness result here is more subtle and requires more discussion and additional notions.

Fix a vocabulary σ. Let Q be a FO^+ query over σ and let *Mat* be relational-to-matrix schema encoding on $\sigma(Q)$. We say that a matrix query $\mathbf{Q}$ *simulates* Q w.r.t. *Mat* if $Mat(\llbracket Q \rrbracket_{db}) = \llbracket \mathbf{Q} \rrbracket(Mat(db))$ for every database *db* consistent with *Mat*. We note again that this definition implies that the input vocabulary and output relation symbol of Q coincides with the input schema and output matrix symbol of $\mathbf{Q}$, respectively. Further, it is equivalent that $\llbracket Q \rrbracket_{db} = Mat^{-1}(\llbracket \mathbf{Q} \rrbracket(Mat(db)))$.

Before stating how to connect FO^+ with sum-MATLANG, we need to overcome the following problem: a FO^+ query can use the same variable within different relational atoms, which can be mapped to matrix symbols of different types. For an illustrative example of this problem, consider the query

$$Q\colon H(x,y) \leftarrow R(x,y), S(y,z)$$

and a relational-to-matrix schema encoding such that *Mat* maps R and S to symbols of type (α,β), H to a symbol of type (β,β), and c and d to α and β, respectively. For a consistent database *db* w.r.t. *Mat*, we could have that R and S are consistent with $c^{db} \times d^{db}$, but H is not consistent with $d^{db} \times d^{db}$ if $d^{db} < c^{db}$. Moreover, *Mat* bounds variable y with different sizes c^{db} and d^{db}. It is then problematic to simulate Q under *Mat* in sum-MATLANG because sum-MATLANG expressions need to be well-typed.

Given the previous discussion, the *well-typedness* definition of a FO^+ formula is necessary. Let *Mat* be a relational-to-matrix schema encoding on σ. Given a FO^+ formula φ over σ and a function τ from $free(\varphi)$ to size symbols in $Mat(\sigma)$, define the rule $Mat \vdash \varphi\colon \tau$ inductively as shown in Figure 1, where $\tau_1 \sim \tau_2$ if and only if $\tau_1(x) = \tau_2(x)$ for every $x \in dom(\tau_1) \cap dom(\tau_2)$.

$$\frac{\texttt{type}(Mat(R)) = (\alpha,\beta)}{Mat \vdash R(x_1,x_2)\colon \{x_1 \mapsto \alpha, x_2 \mapsto \beta\}} \qquad \frac{\texttt{type}(Mat(R)) = (\alpha,1) \text{ or } (1,\alpha)}{Mat \vdash R(x)\colon \{x \mapsto \alpha\}}$$

$$\frac{\texttt{type}(Mat(R)) = (1,1)}{Mat \vdash R()\colon \{\}} \qquad \frac{}{Mat \vdash x \leq c\colon \{x \mapsto Mat(c)\}} \qquad \frac{Mat \vdash \varphi\colon \tau}{Mat \vdash \exists \overline{y}.\varphi\colon \tau|_{free(\varphi)\setminus\overline{y}}}$$

$$\frac{Mat \vdash \varphi_1\colon \tau_1, Mat \vdash \varphi_2\colon \tau_2 \text{ and } \tau_1 \sim \tau_2}{Mat \vdash \varphi_1 \wedge \varphi_2\colon \tau_1 \cup \tau_2} \qquad \frac{Mat \vdash \varphi_1\colon \tau_1, Mat \vdash \varphi_2\colon \tau_2 \text{ and } \tau_1 \sim \tau_2}{Mat \vdash \varphi_1 \vee \varphi_2\colon \tau_1 \cup \tau_2}$$

Figure 1 Well-typedness of FO^+ formulas under relational-to-matrix mapping *Mat*.

We say that φ over σ is *well-typed* w.r.t. *Mat* if there exists such a function τ such that $Mat \vdash \varphi\colon \tau$. Note that if φ is well-typed, then there is a unique τ such that $Mat \vdash \varphi\colon \tau$.

Now, let $Q\colon H(\overline{x}) \leftarrow \varphi$ be a binary FO^+ query over σ and *Mat* be a matrix encoding specification on $\sigma(Q)$. We say that Q is *well-typed* w.r.t. *Mat* if φ is well-typed w.r.t. *Mat* and for τ such that $Mat \vdash \varphi\colon \tau$, we have:

- if $\overline{x} = (x_1, x_2)$, then $\texttt{type}(Mat(H)) = (\tau(x_1), \tau(x_2))$; or
- if $\overline{x} = (x)$, then $\texttt{type}(Mat(H))$ is either $(\tau(x), 1)$ or $(1, \tau(x))$.

We write $Mat \vdash Q\colon \tau$ to indicate that Q is well-typed w.r.t. *Mat*, and τ is the unique function testifying to well-typedness of FO^+ formula of Q. We note that that we can show that the query obtained by Proposition 4 is always well-typed.

The next proposition connects well-typedness with consistency.

▶ **Proposition 5.** *For binary* FO^+ *query* $Q\colon H(\overline{x}) \leftarrow \varphi$ *over a vocabulary* σ*, if* $Mat \vdash Q\colon \tau$ *then for any db consistent with Mat we have:*

- *If* $\overline{x} = (x_1, x_2)$ *then* $[\![Q]\!]_{db}(H)$ *is consistent with dimension* $\tau(x_1)^{db} \times \tau(x_2)^{db}$.
- *If* $\overline{x} = (x)$ *then* $[\![Q]\!]_{db}(H)$ *is consistent with both dimension* $\tau(x)^{db} \times 1$ *and* $1 \times \tau(x)^{db}$.

We have now all the formal machinery to state how to simulate every FO^+ query over relations with a sum-MATLANG query over matrices.

▶ **Proposition 6.** *For every binary* FO^+ *query* Q *over a vocabulary* σ *and every relational-to-matrix schema encoding Mat on* $\sigma(Q)$ *such that* Q *is well typed w.r.t. Mat there exists a* sum-MATLANG *query* $\mathbf{Q}$ *that simulates* Q *w.r.t. Mat.*

Conjunctive Matlang. Taking into account the correspondence between sum-MATLANG and FO^+ established by Propositions 5 and 6, in what follows we say that matrix query language $\mathcal{L}_M \subseteq$ sum-MATLANG and relational language $\mathcal{L}_R \subseteq \mathsf{FO}^+$ are *equivalent* or *equally expressive* if (1) for every matrix query $\mathbf{Q} \in \mathcal{L}_M$ over a matrix schema $\mathcal{S}$ and every matrix-to-relational schema encoding *Rel* on $\mathcal{S}(\mathbf{Q})$ there exists a query $Q \in \mathcal{L}_R$ that simulates $\mathbf{Q}$ w.r.t. *Rel* and is well-typed w.r.t. Rel^{-1}; and (2) for every binary query $Q \in \mathcal{L}_R$ over a vocabulary σ and every relational-to-matrix schema encoding *Mat* such that Q is well-typed w.r.t. *Mat* there exists $\mathbf{Q} \in \mathcal{L}_M$ that simulates Q w.r.t *Mat*.

Let conj-MATLANG be the sum-MATLANG fragment that includes all operations except matrix addition (+). Then we can derive the following characterization of CQs.

▶ **Corollary 7.** conj-MATLANG *and conjunctive queries are equally expressive.*

While this result is a consequence of the connection between sum-MATLANG and FO^+, it provides the basis to explore the fragments of conj-MATLANG that correspond to fragments of CQ, like free-connex or q-hierarchical CQ. We determine these fragments in the next sections.

4 The fragment of free-connex queries

In this section, we specialize the correspondence of Corollary 7 between conj-MATLANG and CQs to *free-connex* CQs [5]. Free-connex CQs are a subset of acyclic CQs that allow efficient enumeration-based query evaluation: in the Boolean semiring and under data complexity they allow to enumerate the query result $[\![Q]\!]_{db}(H)$ of free-connex CQ $Q\colon H(\overline{x}) \leftarrow \varphi$ with *constant delay* after a preprocessing phase that is linear in *db*. In fact, under complexity-theoretic assumptions, the class of CQs that admits constant delay enumeration after linear time preprocessing is precisely the class of free-connex CQs [5].

Acyclic and free-connex CQs. A CQ $Q\colon H(\overline{x}) \leftarrow \exists \overline{y}.a_1 \wedge \cdots \wedge a_n$ is called *acyclic* [7,10,21] if it has a *join-tree*, i.e. an undirected tree $T = (V, E)$ with V the set $\{a_1, \ldots, a_n\}$ of atoms in the body and where for each variable z occurring in Q the set $\{a_i \in V \mid z \in vars(a_i)\}$ induces a connected subtree of T. Note that we consider inequality predicates as unary. Furthermore, Q is *free-connex* [5,11] if it is acyclic and the query Q' obtained by adding the head atom $H(\overline{x})$ to the body of Q is also acyclic. The second condition forbids queries like $H(x, y) \leftarrow \exists z.A(x, z) \wedge B(z, y)$ since when we adjoin the head to the body we get $\exists z.A(x, z) \wedge B(z, y) \wedge H(x, y)$, which is cyclic. We will usually refer to free-connex CQs simply as fc-CQs in what follows.

To identify the sum-MATLANG fragment that corresponds to fc-CQs, we find it convenient to first observe the following correspondence between fc-CQs and $\mathsf{FO}^{\wedge}_2$, the two-variable fragment of $\mathsf{FO}^{\wedge}$. Here, a formula φ in $\mathsf{FO}^{\wedge}$ is said to be in $\mathsf{FO}^{\wedge}_2$ if the set of all variables used in φ (free or bound) is of cardinality at most two. So, $\exists y \exists z.A(x, y) \wedge B(y, z)$ is not in $\mathsf{FO}^{\wedge}_2$, but the equivalent formula $\exists y.\,(A(x, y) \wedge \exists x.B(y, x))$ is. An $\mathsf{FO}^{\wedge}_2$ query is a binary query whose body is an $\mathsf{FO}^{\wedge}_2$ formula. Recall that a query is binary if every relational atom occurring in its body and head have arity at most two.

▶ **Theorem 8.** *Binary free-connex CQs and* $\mathsf{FO}^{\wedge}_2$ *queries are equally expressive.*

We find this a remarkable characterization of the fc-CQs on binary relations that, to the best of our knowledge, it is new. Moreover, this result motivates the fragment of MATLANG that characterizes fc-CQs.

Free-connex MATLANG**.** Define fc-MATLANG to be the class of all MATLANG expressions generated by the grammar:

$$e \quad ::= \quad \mathbf{A} \mid \mathbf{1}^{\alpha} \mid \mathbf{I}^{\alpha} \mid e^T \mid e_1 \times e_2 \mid e_1 \odot e_2 \mid e_1 \cdot v_2 \mid v_1 \cdot e_2$$

where v_1 and v_2 are fc-MATLANG expressions with type $(\alpha, 1)$ or $(1, \alpha)$. In other words, matrix multiplication $e_1 \cdot e_2$ is only allowed when at least one of e_1 or e_2 has a row or column vector type.

Interestingly, we are able to show that $\mathsf{FO}^{\wedge}_2$ and fc-MATLANG are equally expressive.

▶ **Theorem 9.** fc-MATLANG *and* $\mathsf{FO}^{\wedge}_2$ *are equally expressive.*

From Theorem 8 and Theorem 9 we obtain:

▶ **Corollary 10.** fc-MATLANG *and binary free-connex CQs are equally expressive.*

5 The fragment of q-hierarchical queries

We next specialize the correspondence between conj-MATLANG and CQs to *q-hierarchical* CQs [5]. The q-hierarchical CQs form a fragment of the free-connex CQs that, in addition to supporting constant delay enumeration after linear time preprocessing, have the property that every single-tuple update (insertion or deletion) to the input database can be processed in constant time, after which the enumeration of the updated query result can again proceed with constant delay [9].

Q-hierarchical CQs. Let $Q\colon H(\overline{x}) \leftarrow \exists \overline{y}.a_1 \wedge \ldots \wedge a_n$ be a CQ. For every variable x, define $at(x)$ to be the set $\{a_i \mid x \in var(a_i)\}$ of relational atoms that mention x. Note that, contrary to acyclic queries, here we make the distinction between relational atoms and inequalities. Then Q is q-hierarchical if for any two variables x, y the following is satisfied:

1. $at(x) \subseteq at(y)$ or $at(x) \supseteq at(y)$ or $at(x) \cap at(y) = \emptyset$, and
2. if $x \in \overline{x}$ and $at(x) \subsetneq at(y)$ then $y \in \overline{x}$.

For example, $H(x) \leftarrow \exists y.A(x,y) \wedge U(x)$ is q-hierarchical. By contrast, the variant $H(x) \leftarrow \exists y.A(x,y) \wedge U(y)$ is not q-hierarchical, as it violates the second condition. Furthermore, $H(x,y) \leftarrow A(x,y) \wedge U(x) \wedge V(y)$ violates the first condition, and is also not q-hierarchical. Note that all these examples are free-connex. We refer to q-hierarchical CQs simply as qh-CQs.

Q-hierarchical Matlang. The fragment of sum-MATLANG that is equivalent to qh-CQs is a two-layered language where expressions in a higher layer can only be built from the lower layer. This lower layer, called simple-MATLANG, is a fragment of fc-MATLANG defined as:

$$e \;::=\; \mathbf{A} \mid \mathbf{1}^{\alpha} \mid \mathbf{I}^{\alpha} \mid e^{T} \mid e_1 \times e_2 \mid e_1 \odot e_2 \mid e \cdot \mathbf{1}^{\alpha}.$$

Note that in simple-MATLANG matrix multiplication is further restricted to matrix-vector multiplication with the ones vector. Intuitively, all simple-MATLANG expressions can already define q-hierarchical CQs like $H(x) \leftarrow \exists y.A(x,y) \wedge U(x)$, but it cannot define cross-products like $H(x,y) \leftarrow A(x) \wedge B(y)$, which are q-hierarchical. For this reason, we need to enhance simple-MATLANG with the higher layer. Specifically, we define qh-MATLANG as follows:

$$e \;::=\; e_1 \mid e_1 \odot \left(e_2 \cdot (\mathbf{1}^{\alpha})^{T}\right) \mid \left(\mathbf{1}^{\alpha} \cdot e_1\right) \odot e_2 \mid \left(\mathbf{1}^{\alpha} \cdot e_1\right) \odot \left(e_2 \cdot (\mathbf{1}^{\alpha})^{T}\right)$$

where e_1 and e_2 are simple-MATLANG expressions. Note that the subexpressions $\mathbf{1}^{\alpha} \cdot e_1$ and $e_2 \cdot (\mathbf{1}^{\alpha})^{T}$ are valid if e_1 and e_2 have a row and column vector type, respectively. Then, both subexpressions are useful for expanding vector-type expressions into a matrix-type expression.

The qh-MATLANG syntax does not allow expressions like, for example, $\mathbf{1}^{\alpha} \cdot e_1$ where e_1 is a simple-MATLANG expression. Nevertheless, one can define this expression alternatively as $(\mathbf{1}^{\alpha} \cdot e_1) \odot (\mathbf{1}^{\alpha} \cdot (\mathbf{1}^{\alpha})^{T})$. For presentational purposes, we decided to define qh-MATLANG as simple as possible, leaving out some expressions in fc-MATLANG that are not in qh-MATLANG, although an equivalent qh-MATLANG expression defines it.

▶ **Theorem 11.** qh-MATLANG *and binary q-hierarchical CQs are equally expressive.*

6 Efficient evaluation of free-connex and q-hierarchical queries

Now that we have precise connections between subfragments of MATLANG and subfragments of CQ, we can use these connections to derive efficient evaluation algorithms for MATLANG. Unfortunately, to apply the algorithms for CQ, we must first face two problems: (1) the

evaluation algorithms for fc-CQs and qh-CQs are usually restricted to the Boolean semiring, and (2) these algorithms are for CQ without inequalities (comparison atoms). To overcome these problems, we need to revisit the fc-CQs and qh-CQs evaluation problem, generalize the algorithmic results to other semirings (e.g., $\mathbb{R}$), and extend the results when queries have also inequalities. Only then can we derive efficient algorithms for the fragments of fc-MATLANG and qh-MATLANG.

The evaluation setting. In our setting, we consider query evaluation as an enumeration problem. Specifically, let Q be a CQ over vocabulary σ, H its relation symbol in the head, and $\mathcal{K}$ a semiring. We define the evaluation problem $\texttt{Eval}(Q, \sigma, \mathcal{K})$ as follows:

Problem:	$\texttt{Eval}(Q, \sigma, \mathcal{K})$
Input:	A $\mathcal{K}$-database db over σ
Output:	Enumerate $\{(\bar{d}, [\![Q]\!]_{db}(H)(\bar{d})) \mid [\![Q]\!]_{db}(H)(\bar{d}) \neq \mathbb{0}\}$.

In other words, the evaluation problems ask to retrieve the set of all tuples output by Q on db, together with their non-zero annotations. Similarly, we can define the evaluation problem $\texttt{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ for a conj-MATLANG query $\mathbf{Q}$ over a matrix schema $\mathcal{S}$ where we aim to enumerate the non-zero entries of $[\![\mathbf{Q}]\!](\mathcal{I})$ given as input a matrix instance $\mathcal{I}$ over $\mathcal{S}$, represented sparsely as the set of its non-zero entries.

As it is standard in the area, we consider enumeration algorithms on the Random Access Machine (RAM) with uniform cost measure [3]. We assume that semiring values can be represented in $\mathcal{O}(1)$ space and that the semiring operations $\oplus$ and $\odot$ can be evaluated in $\mathcal{O}(1)$ time. We say that $\texttt{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ can be evaluated with *linear-time preprocessing and constant-delay*, if there exists an enumeration algorithm that takes $\mathcal{O}(\|db\|)$ time to preprocess the input database db, and then retrieves each output $(\bar{d}, [\![Q]\!]_{db}(H)(\bar{d}))$ one by one, without repetitions, and with constant-delay per output. Here, the size of database db is defined to be the number of non-zero entries. The same extends to $\texttt{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ as expected. Note that we measure the time and delay in *data complexity* as is standard in the literature [5, 8, 9, 32].

Evaluation of free-connex queries. We are ready to state the algorithmic results for fc-CQ and fc-MATLANG. A semiring $(K, \oplus, \odot, \mathbb{0}, \mathbb{1})$ is *zero-divisor free* if, for all $a, b \in K$, $a \odot b = \mathbb{0}$ implies $a = \mathbb{0}$ or $b = \mathbb{0}$. A zero-divisor free semiring is called a *semi-integral domain* [25]. Note that semirings used in practice, like $\mathbb{B}$, $\mathbb{N}$, and $\mathbb{R}$, are semi-integral domains.

▶ **Theorem 12.** *Let $\mathcal{K}$ be a semi-integral domain. For every free-connex query Q over σ, $\texttt{Eval}(Q, \sigma, \mathcal{K})$ can be evaluated with linear-time preprocessing and constant-delay. In particular, $\texttt{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ can also be evaluated with linear-time preprocessing and constant-delay for every* fc-MATLANG $\mathbf{Q}$ *over $\mathcal{S}$.*

The semi-integral condition is necessary to ensure that zero outputs could only be produced by some zero entries. For instance, consider a semiring $(K, \oplus, \odot, \mathbb{0}, \mathbb{1})$ such that there exist $a, b \in K$ where $a \neq \mathbb{0}$, $b \neq \mathbb{0}$ and $a \odot b = \mathbb{0}$. Further, consider the query $Q : H(x, y) \leftarrow R(x) \wedge S(y)$ over the previous semiring. Let R and S be relation symbols with arity one and db a database over $\sigma = \{R, S\}$ such that $R(1) = a; R(2) = a; S(1) = b$ and $S(2) = b$. Then, the output of $\texttt{Eval}(Q, \sigma, \mathcal{K})$ with input db is empty, although the body can be instantiated in four different ways, all of them producing $\mathbb{0}$ values.

We derive the enumeration algorithm for $\texttt{Eval}(Q, \sigma, \mathcal{K})$ by extending the algorithms in [5] for any semi-integral domain $\mathcal{K}$ and taking care of the inequalities in Q. We derive the enumeration algorithm for $\texttt{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ by reducing to $\texttt{Eval}(Q, \sigma, \mathcal{K})$, using Corollary 10.

To illustrate the utility of Theorem 12, consider the fc-MATLANG query $\mathbf{A} \odot (\mathbf{U} \cdot \mathbf{V}^T)$ where $\mathbf{U}, \mathbf{V}$ are column vectors. When this query is evaluated in a bottom-up fashion, the subexpression $(\mathbf{U} \cdot \mathbf{V}^T)$ will generate partial results of size $\|\mathbf{U}\|\|\mathbf{V}\|$, causing the entire evaluation to be of complexity $\Omega(\|\mathbf{A}\| + \|\mathbf{U}\|\|\mathbf{V}\|)$. By contrast, Theorem 12 tells us that we may evaluate the query in time $\mathcal{O}(\|\mathbf{A}\| + \|\mathbf{U}\| + \|\mathbf{V}\|)$.

For lower bounds, we can also extend the results in [5] and [8] for the relational setting under standard complexity assumptions. As in previous work, our lower bounds are for CQ *without self-joins* and we need some additional restrictions for inequalities. Specifically, we say that an inequality $x \leq c$ in Q is *covered*, if there exists a relational atom in Q that mention x. We say that Q is *constant-disjoint* if (i) for all covered inequalities $x \leq c$ we have $c \neq 1$, and (ii) for all pairs $(x \leq c, y \leq d)$ in Q of covered inequality $x \leq c$ and non-covered inequality $y \leq d$ if $c = d$ then $y \notin \mathit{free}(Q)$. In other words, if a constant symbol other than 1 occurs in both a covered and non-covered inequality in Q, then it occurs with a bound variable in the non-covered inequality.

A semiring $\mathcal{K} = (K, \oplus, \odot, \mathbb{0}, \mathbb{1})$ is *zero-sum free* [27, 28] if for all $a, b \in K$ it holds that $a \oplus b = \mathbb{0}$ implies $a = b = \mathbb{0}$. We are ready to extend the lower bound in [5, 8] as follows.

▶ **Theorem 13.** *Let Q be a CQ over σ without self-joins and constant-disjoint. Let $\mathcal{K}$ be a semiring such that the subsemiring generated by $\mathbb{0}_{\mathcal{K}}$ and $\mathbb{1}_{\mathcal{K}}$ is zero-sum free. If $\mathtt{Eval}(Q, \sigma, \mathcal{K})$ can be evaluated with linear-time preprocessing and constant-delay, then Q is free-connex, unless either the Sparse Boolean Matrix Multiplication, the Triangle Detection, or the $(k, k+1)$-Hyperclique conjecture is false.*

It is important to note that most semirings used in practice, like $\mathbb{B}$, $\mathbb{N}$, and $\mathbb{R}$, are such that the subsemiring generated by $\mathbb{0}_{\mathcal{K}}$ and $\mathbb{1}_{\mathcal{K}}$ is zero-sum free. Furthermore, the Sparse Boolean Matrix Multiplication conjecture, the Triangle Detection conjecture, and the $(k, k+1)$-Hyperclique conjecture are standard complexity assumptions used by previous works [5, 8] (see the formal statements in the full version [38]).

Unfortunately, given the asymmetry between the relational and matrix settings, the lower bounds do not immediately transfer from the relational to the matrix setting. Specifically, we need a syntactical restriction for conj-MATLANG that implies the constant-disjointedness restriction in the translation of Theorem 8. Intuitively, this happens when both dimensions of a conj-MATLANG query $\mathbf{H} := e$ are fixed by matrix symbols in $\mathcal{S}$. For example, the expressions $\mathbf{A} \odot (\mathbf{U} \cdot \mathbf{V}^T)$ and $\Sigma \mathbf{v}.\, \mathbf{A} \cdot \mathbf{v}$ have both dimensions (i.e., row and column) fixed by $\mathbf{A}$ where $\mathbf{U}, \mathbf{V}$ are column vectors. Instead, the expression $\mathbf{U} \cdot (\mathbf{1}^\alpha)^T$ does not, since its column dimension depends on the value assigned for α and is not necessarily fixed by $\mathbf{U}$. Formally, $\mathtt{FixDim}(e)$ is the set of *fixed dimensions* of a conj-MATLANG expression e and it is inductively defined as follows:

$$
\begin{aligned}
\mathtt{FixDim}(\mathbf{A}) &:= \{0, 1\} \\
\mathtt{FixDim}(\mathbf{v}) &:= \{1\} \\
\mathtt{FixDim}(e^T) &:= \{(i+1) \mod 2 \mid i \in \mathtt{FixDim}(e)\} \\
\mathtt{FixDim}(e_1 \cdot e_2) &:= (\mathtt{FixDim}(e_1) \setminus \{1\}) \cup (\mathtt{FixDim}(e_2) \setminus \{0\}) \\
\mathtt{FixDim}(e_1 \times e_2) &:= \mathtt{FixDim}(e_2) \\
\mathtt{FixDim}(e_1 \odot e_2) &:= \mathtt{FixDim}(e_1) \cup \mathtt{FixDim}(e_2) \\
\mathtt{FixDim}(\Sigma \mathbf{v}.e) &:= \mathtt{FixDim}(e).
\end{aligned}
$$

An expression e has *guarded dimensions* if $\mathtt{FixDim}(e) = \{0, 1\}$.

The former is straightforwardly extended to a conj-MATLANG query $\mathbf{Q}$, where $\mathtt{FixDim}(\mathbf{Q})$ stands for $\mathtt{FixDim}(e)$. The following lower bound is now attainable.

▶ **Corollary 14.** *Let* $\mathbf{Q}$ *be a* conj-MATLANG *query over* $\mathcal{S}$ *such that* $\mathbf{Q}$ *does not repeat matrix symbols and* $\mathbf{Q}$ *has guarded dimensions. Let* $\mathcal{K}$ *be a semiring such that the subsemiring generated by* $\mathbb{0}_{\mathcal{K}}$ *and* $\mathbb{1}_{\mathcal{K}}$ *is zero-sum free. If* $\texttt{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ *can be evaluated with linear-time preprocessing and constant-delay, then* $\mathbf{Q}$ *is equivalent to a* fc-MATLANG *query, unless either the Sparse Boolean Matrix Multiplication, the Triangle Detection, or the* $(k, k+1)$*-Hyperclique conjecture is false.*

The dynamic evaluation setting. We move now to the dynamic query evaluation both in the relational and matrix scenarios. Specifically, we consider the following set of updates. Recall that $\mathcal{K} = (K, \oplus, \odot, \mathbb{0}, \mathbb{1})$ is a semiring and σ a vocabulary.

- A single-tuple *insertion* (over $\mathcal{K}$ and σ) is an operation $u = \texttt{insert}(R, \overline{d}, k)$ with $R \in \sigma$, $\overline{d}$ a tuple of arity $ar(R)$, and $k \in K$. When applied to a database db it induces the database $db + u$ that is identical to db, but $R^{db+u}(\overline{d}) = R^{db}(\overline{d}) \oplus k$.
- A single-tuple *deletion* (over $\mathcal{K}$ and σ) is an expression $u = \texttt{delete}(R, \overline{d})$ with $R \in \sigma$ and $\overline{d}$ a tuple of arity $ar(R)$. When applied to a database db it induces the database $db + u$ that is identical to db, but $R^{db+u}(\overline{d}) = \mathbb{0}$.

Notice that if every element in $\mathcal{K}$ has an additive inverse (i.e., $\mathcal{K}$ is a ring), one can simulate a deletion with an insertion. However, if this is not the case (e.g., $\mathbb{B}$ or $\mathbb{N}$), then a single-tuple deletion is a necessary operation.

We define the dynamic query evaluation problem $\texttt{DynEval}(Q, \sigma, \mathcal{K})$ as the extension of $\texttt{Eval}(Q, \sigma, \mathcal{K})$ under the set of updates U of all single-tuple insertions and deletions over $\mathcal{K}$ and σ. We say that $\texttt{DynEval}(Q, \sigma, \mathcal{K})$ can be evaluated dynamically with *constant-time update and constant-delay*, if $\texttt{Eval}(Q, \sigma, \mathcal{K})$ can be evaluated with linear-time preprocessing and constant delay, and, moreover, for every update $u \in U$, it takes constant-time to update the state of the algorithm from db to $db + u$ so that, immediately after, we can retrieve each output $(\overline{d}, [\![Q]\!]_{db+u}(H)(\overline{d}))$ one by one, without repetitions, and with constant-delay per output. Similarly, we define the dynamic query evaluation problem $\texttt{DynEval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ of $\texttt{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ for the matrix setting, requiring the same dynamic guarantees.

Note that updates allow to modify the contents of relations, but not of constant symbols. Similarly, in the linear algebra setting updates only affect matrix entry values, not matrix dimensions. An interesting line of future work is to consider dimension updates, which correspond to allow updates of constant symbols in CQs.

Evaluation of q-hierarchical queries. Similar than for free-connex queries, we can provide dynamic evaluation algorithms for qh-CQ and qh-MATLANG queries. However, for this dynamic setting, we require some additional algorithmic assumptions over the semiring. Let $\mathcal{K} = (K, \oplus, \odot, \mathbb{0}, \mathbb{1})$ be a semiring and M be the set of all multisets of K. For any $k \in K$ and $m \in M$, define $\texttt{ins}(k, m)$ and $\texttt{del}(k, m)$ to be the multisets resulting from inserting or deleting k from m, respectively. Then we say that $\mathcal{K}$ is *sum-maintainable* if there exists a data structure $\mathcal{D}$ to represent multisets of K such that the empty set $\emptyset$ can be built in constant time, and if $\mathcal{D}$ represents $m \in M$ then: **(1)** the value $\bigoplus_{k \in m} k$ can always be computed from $\mathcal{D}$ in constant time; **(2)** a data structure that represents $\texttt{ins}(k, m)$ can be obtained from $\mathcal{D}$ in constant time; and **(3)** a data structure that represents $\texttt{del}(k, m)$ can be obtained from $\mathcal{D}$ in constant time. One can easily notice that if each element of $\mathcal{K}$ has an additive inverse (i.e., $\mathcal{K}$ is a ring), then $\mathcal{K}$ is sum-maintainable, like $\mathbb{R}$. Other examples of sum-maintainable semirings (without additive inverses) are $\mathbb{B}$ and $\mathbb{N}$.

▶ **Theorem 15.** *Let* $\mathcal{K}$ *be a sum-maintainable semi-integral domain. For every q-hierarchical CQ* Q*,* $\texttt{DynEval}(Q, \sigma, \mathcal{K})$ *can be evaluated dynamically with constant-time update and constant-*

delay. In particular, $\texttt{DynEval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ *can also be evaluated dynamically with constant-time update and constant-delay for every* qh-MATLANG $\mathbf{Q}$ *over* $\mathcal{S}$*.*

Similar than for free-connex CQ, we can extend the lower bound in [9] when the subsemiring generated by $\mathbb{0}_{\mathcal{K}}$ and $\mathbb{1}_{\mathcal{K}}$ is zero-sum free, by assuming the Online Boolean Matrix-Vector Multiplication (OMv) conjecture [29].

▶ **Theorem 16.** *Let* Q *be a CQ over* σ *without self-joins and constant-disjoint. Let* $\mathcal{K}$ *be a semiring such that the subsemiring generated by* $\mathbb{0}_{\mathcal{K}}$ *and* $\mathbb{1}_{\mathcal{K}}$ *is zero-sum free. If* $\texttt{DynEval}(Q, \sigma, \mathcal{K})$ *can be evaluated dynamically with constant-time update and constant-delay, then* Q *is q-hierarchical, unless the OMv conjecture is false.*

This transfers to conj-MATLANG, similarly to the lower bound in the free-connex case.

▶ **Corollary 17.** *Let* $\mathbf{Q}$ *be a* conj-MATLANG *query over* $\mathcal{S}$ *such that* $\mathbf{Q}$ *does not repeat matrix symbols and* $\mathbf{Q}$ *has guarded dimensions. Let* $\mathcal{K}$ *be a semiring such that the subsemiring generated by* $\mathbb{0}_{\mathcal{K}}$ *and* $\mathbb{1}_{\mathcal{K}}$ *is zero-sum free. If* $\texttt{DynEval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ *can be evaluated dynamically with constant-time update and constant-delay, then* $\mathbf{Q}$ *is equivalent to a* qh-MATLANG *query, unless the OMv conjecture is false.*

7 Conclusions and future work

In this work, we isolated the subfragments of conj-MATLANG that admit efficient evaluation in both static and dynamic scenarios. We found these algorithms by making the correspondence between CQ and MATLANG, extending the evaluation algorithms for free-connex and q-hierarchical CQ, and then translating these algorithms to the corresponding subfragments, namely, fc-MATLANG and qh-MATLANG. To the best of our knowledge, this is the first work that characterizes subfragments of linear algebra query languages that admit efficient evaluation. Moreover, this correspondence improves our understanding of its expressibility.

Regarding future work, a relevant direction is to extend fc-MATLANG and qh-MATLANG with disjunction, namely, matrix summation. This direction is still an open problem even for CQ with union [14]. Another natural extension is to add point-wise functions and understand how they affect expressibility and efficient evaluation. Finally, improving the lower bounds to queries without self-join would be interesting, which is also an open problem for CQ [8, 15].

References

1 Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In Kimberly Keeton and Timothy Roscoe, editors, *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, pages 265–283. USENIX Association, 2016. URL: `https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi`.

2 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: `http://webdam.inria.fr/Alice/`.

3 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

4 Michael J. Anderson, Shaden Smith, Narayanan Sundaram, Mihai Capota, Zheguang Zhao, Subramanya Dulloor, Nadathur Satish, and Theodore L. Willke. Bridging the gap between HPC and big data frameworks. *Proc. VLDB Endow.*, 10(8):901–912, 2017. URL: `http://www.vldb.org/pvldb/vol10/p901-anderson.pdf`, `doi:10.14778/3090163.3090168`.

5 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 208–222. Springer, 2007. doi:10.1007/978-3-540-74915-8_18.

6 Pablo Barceló, Nelson Higuera, Jorge Pérez, and Bernardo Subercaseaux. On the expressiveness of LARA: A unified language for linear and relational algebra. In Carsten Lutz and Jean Christoph Jung, editors, *23rd International Conference on Database Theory, ICDT 2020, March 30-April 2, 2020, Copenhagen, Denmark*, volume 155 of *LIPIcs*, pages 6:1–6:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. URL: https://doi.org/10.4230/LIPIcs.ICDT.2020.6, doi:10.4230/LIPICS.ICDT.2020.6.

7 Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, 1983. doi:10.1145/2402.322389.

8 Christoph Berkholz, Fabian Gerhardt, and Nicole Schweikardt. Constant delay enumeration for conjunctive queries: a tutorial. *ACM SIGLOG News*, 7(1):4–33, 2020. doi:10.1145/3385634.3385636.

9 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts, editors, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 303–318. ACM, 2017. doi:10.1145/3034786.3034789.

10 Philip A. Bernstein and Nathan Goodman. Power of natural semijoins. *SIAM J. Comput.*, 10(4):751–771, 1981. doi:10.1137/0210059.

11 Johann Brault-Baron. *De la pertinence de l'énumération : complexité en logiques propositionnelle et du premier ordre. (The relevance of the list: propositional logic and complexity of the first order)*. PhD thesis, University of Caen Normandy, France, 2013. URL: https://tel.archives-ouvertes.fr/tel-01081392.

12 Robert Brijder, Floris Geerts, Jan Van den Bussche, and Timmy Weerwag. On the expressive power of query languages for matrices. *ACM Trans. Database Syst.*, 44(4):15:1–15:31, 2019. doi:10.1145/3331445.

13 Robert Brijder, Marc Gyssens, and Jan Van den Bussche. On matrices and k-relations. In Andreas Herzig and Juha Kontinen, editors, *Foundations of Information and Knowledge Systems - 11th International Symposium, FoIKS 2020, Dortmund, Germany, February 17-21, 2020, Proceedings*, volume 12012 of *Lecture Notes in Computer Science*, pages 42–57. Springer, 2020. doi:10.1007/978-3-030-39951-1_3.

14 Nofar Carmeli and Markus Kröll. On the enumeration complexity of unions of conjunctive queries. *ACM Trans. Database Syst.*, 46(2):5:1–5:41, 2021. doi:10.1145/3450263.

15 Nofar Carmeli and Luc Segoufin. Conjunctive queries with self-joins, towards a fine-grained enumeration complexity analysis. In Floris Geerts, Hung Q. Ngo, and Stavros Sintos, editors, *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2023, Seattle, WA, USA, June 18-23, 2023*, pages 277–289. ACM, 2023. doi:10.1145/3584372.3588667.

16 Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. Comput. Log.*, 8(4):21, 2007. doi:10.1145/1276920.1276923.

17 Arnaud Durand, Nicole Schweikardt, and Luc Segoufin. Enumerating answers to first-order queries over databases of low degree. In Richard Hull and Martin Grohe, editors, *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 121–131. ACM, 2014. doi:10.1145/2594538.2594539.

18 Idan Eldar, Nofar Carmeli, and Benny Kimelfeld. Direct access for answers to conjunctive queries with aggregation. *CoRR*, abs/2303.05327, 2023. URL: https://doi.org/10.48550/arXiv.2303.05327, arXiv:2303.05327, doi:10.48550/ARXIV.2303.05327.

19 Tarek Elgamal, Shangyu Luo, Matthias Boehm, Alexandre V. Evfimievski, Shirish Tatikonda, Berthold Reinwald, and Prithviraj Sen. SPOOF: sum-product optimization and operator fusion for large-scale machine learning. In *8th Biennial Conference on Innovative Data Systems Research, CIDR 2017, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*. www.cidrdb.org, 2017. URL: http://cidrdb.org/cidr2017/papers/p3-elgamal-cidr17.pdf.

20 Ahmed Elgohary, Matthias Boehm, Peter J. Haas, Frederick R. Reiss, and Berthold Reinwald. Scaling machine learning via compressed linear algebra. *SIGMOD Rec.*, 46(1):42–49, 2017. doi:10.1145/3093754.3093765.

21 Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3):514–550, 1983. doi:10.1145/2402.322390.

22 Floris Geerts. On the expressive power of linear algebra on graphs. In Pablo Barceló and Marco Calautti, editors, *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal*, volume 127 of *LIPIcs*, pages 7:1–7:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. URL: https://doi.org/10.4230/LIPIcs.ICDT.2019.7, doi:10.4230/LIPICS.ICDT.2019.7.

23 Floris Geerts, Thomas Muñoz, Cristian Riveros, and Domagoj Vrgoc. Expressive power of linear algebra query languages. In Leonid Libkin, Reinhard Pichler, and Paolo Guagliardo, editors, *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2021, Virtual Event, China, June 20-25, 2021*, pages 342–354. ACM, 2021. doi:10.1145/3452021.3458314.

24 Floris Geerts and Juan L. Reutter. Expressiveness and approximation properties of graph neural networks. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL: https://openreview.net/forum?id=wIzUeM3TAU.

25 Jonathan S Golan. *Semirings and their Applications*. Springer Science & Business Media, 2013.

26 Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In Leonid Libkin, editor, *Proceedings of the 26th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 2007, Beijing, China, June 11-13, 2007*, pages 31–40. ACM, 2007. doi:10.1145/1265530.1265535.

27 Udo Hebisch and Hanns Joachim Weinert. *Semirings: algebraic theory and applications in computer science*, volume 5. World Scientific, 1998.

28 Udo Hebisch and Hans Joachim Weinert. Semirings and semifields. *Handbook of Algebra*, 1:425–462, 1996.

29 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 21–30. ACM, 2015. doi:10.1145/2746539.2746609.

30 Botong Huang, Shivnath Babu, and Jun Yang. Cumulon: optimizing statistical data analysis in the cloud. In Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 1–12. ACM, 2013. doi:10.1145/2463676.2465273.

31 Muhammad Idris, Martín Ugarte, and Stijn Vansummeren. The dynamic yannakakis algorithm: Compact and efficient query processing under updates. In Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu, editors, *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 1259–1274. ACM, 2017. doi:10.1145/3035918.3064027.

32 Muhammad Idris, Martín Ugarte, Stijn Vansummeren, Hannes Voigt, and Wolfgang Lehner. General dynamic yannakakis: conjunctive queries with theta joins under updates. *VLDB J.*, 29(2-3):619–653, 2020. URL: https://doi.org/10.1007/s00778-019-00590-9, doi:10.1007/S00778-019-00590-9.

33 Dimitrije Jankov, Shangyu Luo, Binhang Yuan, Zhuhua Cai, Jia Zou, Chris Jermaine, and Zekai J. Gao. Declarative recursive computation on an RDBMS: or, why you should use a database for distributed machine learning. *SIGMOD Rec.*, 49(1):43–50, 2020. doi:10.1145/3422648.3422659.

34 Konstantinos Kanellopoulos, Nandita Vijaykumar, Christina Giannoula, Roknoddin Azizi, Skanda Koppula, Nika Mansouri-Ghiasi, Taha Shahroodi, Juan Gómez-Luna, and Onur Mutlu. SMASH: co-designing software compression and hardware-accelerated indexing for efficient sparse matrix operations. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2019, Columbus, OH, USA, October 12-16, 2019*, pages 600–614. ACM, 2019. doi:10.1145/3352460.3358286.

35 Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Trade-offs in static and dynamic evaluation of hierarchical queries. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, pages 375–392. ACM, 2020. doi:10.1145/3375395.3387646.

36 Wojciech Kazana and Luc Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. In Richard Hull and Wenfei Fan, editors, *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA, June 22 - 27, 2013*, pages 297–308. ACM, 2013. doi:10.1145/2463664.2463667.

37 Shangyu Luo, Zekai J. Gao, Michael N. Gubanov, Luis Leopoldo Perez, and Christopher M. Jermaine. Scalable linear algebra on a relational database system. *SIGMOD Rec.*, 47(1):24–31, 2018. doi:10.1145/3277006.3277013.

38 Thomas Muñoz, Cristian Riveros, and Stijn Vansummeren. Enumeration and updates for conjunctive linear algebra queries through expressibility. *CoRR*, abs/2310.04118, 2023. URL: https://doi.org/10.48550/arXiv.2310.04118, arXiv:2310.04118, doi:10.48550/ARXIV.2310.04118.

39 Nicole Schweikardt, Luc Segoufin, and Alexandre Vigny. Enumeration for FO queries over nowhere dense graphs. In Jan Van den Bussche and Marcelo Arenas, editors, *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2018, Houston, TX, USA, June 10-15, 2018*, pages 151–163. ACM, 2018. doi:10.1145/3196959.3196971.

40 Amir Shaikhha, Mohammed Elseidy, Stephan Mihaila, Daniel Espino, and Christoph Koch. Synthesis of incremental linear algebra programs. *ACM Trans. Database Syst.*, 45(3):12:1–12:44, 2020. doi:10.1145/3385398.

41 Yisu Remy Wang, Shana Hutchison, Dan Suciu, Bill Howe, and Jonathan Leang. SPORES: sum-product optimization via relational equality saturation for large scale linear algebra. *Proc. VLDB Endow.*, 13(11):1919–1932, 2020. URL: http://www.vldb.org/pvldb/vol13/p1919-wang.pdf.

42 Fan Yang, Yuzhen Huang, Yunjian Zhao, Jinfeng Li, Guanxian Jiang, and James Cheng. The best of both worlds: Big data programming with both productivity and performance. In Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu, editors, *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 1619–1622. ACM, 2017. doi:10.1145/3035918.3058735.