Logical Methods in Computer Science
Volume 22, Issue 1, 2026, pp. 1:1–1:70
https://lmcs.episciences.org/

Submitted Jan. 27, 2025
Published Jan. 01, 2026

ENUMERATION AND UPDATES FOR CONJUNCTIVE LINEAR ALGEBRA QUERIES THROUGH EXPRESSIBILITY

THOMAS MUÑOZ [a], CRISTIAN RIVEROS [b], AND STIJN VANSUMMEREN [a]

[a] UHasselt, Belgium
e-mail address: thomas.munozserrano@uhasselt.be, stijn.vansummeren@uhasselt.be

[b] PUC Chile, Chile
e-mail address: cristian.riveros@uc.cl

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.

1. Introduction

Linear algebra forms the backbone of modern data analytics, as most machine learning algorithms are coded as sequences of matrix operations [ABC$^+$16, ELB$^+$17, ASS$^+$17, EBH$^+$17, YHZ$^+$17]. In practice, linear algebra programs operate over matrices with millions of entries. Therefore, efficient evaluation of linear algebra programs is a relevant challenge for data management systems which has attracted research attention with several proposals in the area [HBY13, KVG$^+$19, WHS$^+$20, JLY$^+$20, LGG$^+$18].

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 [SEM$^+$20, BHPS20, BGdBW19, BGdB20, GMRV21]. One such proposal is MATLANG [BGdBW19], 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

Key words and phrases: Query evaluation, conjunctive queries, linear algebra, enumeration algorithms.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.46298/LMCS-22(1:1)2026

of most linear algebra operations [GMRV21]. 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 [BGS20, BDG07, Bra13, IUV17, KNOZ20, SSV18, KS13, DG07, DSS14].

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 [BDG07, BKS17, BGS20, IUV17]. 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 [GMRV21], 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, which is a natural MATLANG subfragment that we show to be equally expressive as *free-connex CQ* [BDG07], 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* [BKS17, IUV17], 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 [BKS17].

This article is further organized as follows. Section 2 adresses all definitions. Section 3 describes how to go from a relational setting to a linear algebra setting and vice versa. In Section 4 we show sum-MATLANG and positive first-order logic are equally expressive. Section 5 defines free-connex CQs and characterizes binary free-connex CQs with the conjunctive two variable fragment of positive first order logic. Afterwards, fc-MATLANG is defined in Section 6. The definition of q-hierarchical CQs is located in Section 7 and in that matter binary q-hierarchical CQs are shown to be equally expressive as hierarchical queries of the conjunctive two variable fragment of positive first order logic. Subsequently in Section 8, qh-MATLANG is defined. Later, Section 9 defines the enumeration problem and states known upper and lower bounds on the Boolean semiring for free-connex CQs. In Section 10 the upper and lower bounds are extended to general semirings for free-connex CQs which use inequality atoms. Section 11 defines the dynamic enumeration problem and known upper and lower bounds on the Boolean semiring for q-hierarchical CQs; and illustrates how to extend these bounds to general semirings and for q-hierarchical CQs which use inequality atoms. We end by presenting some conclusions and future work in Section 12.

1.1. **New material.** Our results have previously been published in the 27th International Conference on Database Theory, ICDT 2024. This article is an extended and complete version containing full proofs of all formal statements omitted in the conference version. Moreover, several notions and proofs were revisited to improve the presentation and shorten the article. Specifically, we introduce a *prenex normal form* for conj-MATLANG in Section 4, which resulted in being especially useful for several proofs, in particular, for the proof that sum-MATLANG and positive first-order logic are equally expressive. Furthermore, these changes in the main proof required a new presentation of other results, like the lower bound statements of fc-MATLANG and qh-MATLANG in Section 10 and Section 11, respectively.

1.2. **Related work.** In addition to the work that has already been cited above, the following work is relevant. Brijder et al. [BGdB20] 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. [GMRV21] previously established a correspondence between sum-MATLANG and FO^+. However, as we illustrate in Section 3.1 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. [ECK24] 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 distinction 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 $\mathsf{FO}^\wedge_k$, 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 $\mathsf{FO}^\wedge_2$. Similarly, Geerts and Reutter [GR22] 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 Appendix E.1 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 $\mathsf{FO}^\wedge_3$, not $\mathsf{FO}^\wedge_2$ as we show here.

Several proposals [HBY13, KVG+19, WHS+20, JLY+20, LGG+18] 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., $\mathsf{MATLANG}$) with such efficiency guarantees.

2. Preliminaries

In this section we recall the main definitions of $\mathsf{MATLANG}$, 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; $(\alpha, 1)$ 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 [GMRV21] over $\mathcal{S}$ is defined by the 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 their *type* $\mathtt{type}(e)$ is correctly defined as follows:

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

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. We say that an expression e is a *sentence* if e does not have free vector variables. To evaluate expressions that are not sentences, 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:

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

In the last line, it is assumed that $\mathbf{v}\colon (\gamma,1)$, and $\mathbf{b}^n_1, \mathbf{b}^n_2, \ldots, \mathbf{b}^n_n$ denote the n-dimensional 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.

Note that if e is a sentence, then $[\![e]\!]$ is independent of the vector valuation μ in the sense that $[\![e]\!](\mathcal{I},\mu) = [\![e]\!](\mathcal{I},\mu')$ for all vector valuations μ and μ'. We therefore sometimes also simply write $[\![e]\!](\mathcal{I})$ instead of $[\![e]\!](\mathcal{I},\mu)$ when e is a sentence.

Example 2.1. 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 2.2. Let $\mathcal{S}$ and $\mathcal{I}$ be as in Example 2.1. Let $\mathbf{v} \in \mathcal{V}$ where $\mathbf{v}\colon (\gamma,1)$. The expression $\Sigma\mathbf{v}.\mathbf{A}$ is well-typed and

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

Fragments of Sum-Matlang. We recall here two fragments of sum-MATLANG that will be important for the paper. First, we define conj-MATLANG as the sum-MATLANG fragment that includes all operations except matrix addition (+). Second, we define MATLANG 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 language as introduced in [BGdBW19] included an operator for *vector diagonalization*. This operator can be simulated using the $\mathbf{I}^\alpha$-operator,

and conversely $\mathbf{I}^\alpha$ can be simulated using vector diagonalization. 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 [BGdBW19, GMRV21], the syntax of MATLANG was more generally parameterized by a family of n-ary functions that could be pointwise applied. Similarly to [BGdB20, Gee19] 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 $\mathbf{H} := e$ where e is a well-typed sum-MATLANG sentence, $\mathbf{H}$ is a "fresh" matrix symbol that does not occur in $\mathcal{S}$, and $\mathtt{type}(\mathbf{H}) = \mathtt{type}(e)$. 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 $\mathcal{S} \cup \{\mathbf{H}\}$.

The former notion of query is extended to any fragment of sum-MATLANG in a natural manner, e.g., a conj-MATLANG query is a sum-MATLANG query $\mathbf{H} := e$ where e is a well-typed conj-MATLANG expression.

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 an 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. An *equality atom* is of the form $x = y$ with x, y variables. A *positive first order logic formula* (FO^+ formula) over a vocabulary σ is an expression generated by the following grammar:

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

where R and c range over relations and constants in σ, respectively. The notions of *free and bound variables* are defined as usual in FO. We denote the set of free variables of φ by $free(\varphi)$ and the multiset of atoms occurring in φ by $at(\varphi)$.

We evaluate formulas over K-relations as follows using the well-known semiring semantics [GKT07]. 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:

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

The *support* of the mapping $[\![\varphi]\!]_{db}$ is the set of valuations that it maps to a nonzero annotation. As the reader may notice, it is possible that $[\![\varphi]\!]_{db}$ has infinite support. This situation is remeniscent of classical FO (evaluated on the Boolean semiring) which may also define infinite relations. To circumvent this problem, we restrict ourselves to *safe* formulas [AHV95], which in classical FO ensures that output relations are finite. To define safety, we first define the set $\mathsf{rr}(\varphi) \subseteq \mathit{free}(\varphi)$ of *range restricted* variables of a formula:

$$
\begin{aligned}
&\mathsf{rr}(R(x_1, \ldots, x_n)) = \{x_1, \ldots, x_n\} \qquad \mathsf{rr}(x \leq c) = \{x\} \qquad \mathsf{rr}(x = y) = \emptyset \\
&\mathsf{rr}(\varphi_1 \wedge \varphi_2) = \mathsf{rr}(\varphi_1) \cup \mathsf{rr}(\varphi_2) \quad \mathsf{rr}(\varphi_1 \vee \varphi_2) = \mathsf{rr}(\varphi_1) \cap \mathsf{rr}(\varphi_2) \quad \mathsf{rr}(\exists y.\varphi) = \mathsf{rr}(\varphi) \setminus \{y\}.
\end{aligned}
$$

An FO^+ formula φ is *safe* if it satisfies the following conditions:

(1) every occurrence of an equality atom $x = y$ is a part of a subformula of φ of the form $\psi \wedge (x = y)$ or $(x = y) \wedge \psi$ where at least one of $\{x, y\}$ is in $\mathsf{rr}(\psi)$;
(2) $\varphi_1 \vee \varphi_2$ is only used when φ_1 and φ_2 have the same set of free variables.

Proposition 2.3. *Every safe FO^+ formula has finite support.*

The proof is remeniscent of that of classical FO; we therefore delegate it to Appendix A. For the rest of this paper, we restrict to safe formulas. Hence when we say "formula" we mean "safe formula", and similarly for queries.

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 $ar(H) = k$. 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

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

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 $\sigma \cup \{H\}$.

Formula and query equivalence. Two FO^+ formulas φ_1 and φ_2 are said to be *equivalent*, denoted $\varphi_1 \equiv \varphi_2$ if $[\![\varphi_1]\!]_{db} = [\![\varphi_2]\!]_{db}$ for every database. Note that because $[\![\varphi_1]\!]_{db}$ is a set of mappings, all with domain $\mathit{free}(\varphi_1)$, and because the same holds for $[\![\varphi_2]\!]_{db}$, we neccesarily have $\mathit{free}(\varphi_1) = \mathit{free}(\varphi_2)$ when $\varphi_1 \equiv \varphi_2$. Similarly, two FO^+ queries Q_1 and Q_2 are equivalent, denoted $Q_1 \equiv Q_2$ if $[\![Q_1]\!]_{db} = [\![Q_2]\!]_{db}$ for every database. This implies that they have the same head but the variables occurring in the heads need not be the same.

Note that because we have fixed $\mathcal{K}$ arbitrarily, formula and query equivalence cannot use the concrete interpretation of the operators in $\mathcal{K}$ operators, and must hence hold for every choice of $\mathcal{K}$.

Fragments of FO^+. 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., $Q : H(\overline{x}) \leftarrow \exists \overline{y}.a_1 \wedge \cdots \wedge a_n$ where $a_1, \ldots, a_n$ are relational, comparison, or equality 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^+ formula is binary if every relational atom occurring in it has arity at most two and it has at most two free variables. Similarly, 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 sum-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

In this section, we lay the grounds to revisit and generalize the connection between sum-MATLANG and FO^+. Towards this goal, we introduce next all the formal machinery necessary to provide translations between the two query languages that work for any matrix schema, are symmetric, and ensure that matrices are encoded as databases of linear

size. These translations are specified in Section 4. We start by discussing why we need to revisit this formal machinery to then continue presenting the formal details.

3.1. **The need to revisit the characterization of Sum-Matlang.** In [GMRV21], the authors already propose a way of translating between sum-MATLANG and FO^+. Next, we summarize how the translations that we propose in this paper improve over those of [GMRV21], justifying why we need to revisit this characterization.

In [GMRV21], the translation from sum-MATLANG to FO^+ goes as follows. It is assumed that on the FO^+ side there is a unary relation R_α for each size symbol α and that, in the database db simulating the sum-MATLANG instance $\mathcal{I}$ we have $R_\alpha^{db}(i) = 1$ for all $1 \leq i \leq \alpha^{\mathcal{I}}$. Note that, consequently, the size of db is larger than that of $\mathcal{I}$: it take unit space to store $\alpha^{\mathcal{I}}$ in $\mathcal{I}$ while it takes $\alpha^{\mathcal{I}}$ space in db. Therefore, $\| db \|$ (the size of db) is certainly not linear in the size of $\mathcal{I}$. Since we are interested in evaluating sum-MATLANG and MATLANG expressions with linear time preprocessing by reducing to the relational case, this encoding of matrix instances as databases is hence unsuitable. We resolve this in our work by adding inequality atoms to FO^+, and adopting a representation of matrix instances by databases that is linear in size.

Conversely, in [GMRV21], the translation from FO^+ to sum-MATLANG only works when the input relations to the FO^+ query represent square matrices of type (α, α), or vectors with type $(\alpha, 1)$ or $(1, \alpha)$, for a fixed single size symbol α such that $\alpha^{db} = n$ where $\{d_1, \ldots, d_n\}$ is the *active domain* of db. This assumption implies that if one translates a sum-MATLANG query into FO^+, optimizes the FO^+ query, and then translates back, one obtains a sum-MATLANG query whose input schema is different than that of the original query. This phenomemon is illustrated by means of the scenario below. By contrast, we establish a more general correspondence, parametrized by schema encodings, that preserves the original input schema in such a round-trip translation.

Consider for example the schema $\mathcal{S} = \{\mathbf{A}\}$ with $\mathbf{A}\colon (\alpha, \beta)$. Let the instance $\mathcal{I}$ over $\mathcal{S}$ be such that $\alpha^{\mathcal{I}} = 3$, $\beta^{\mathcal{I}} = 2$, and:

$$\mathbf{A}^{\mathcal{I}} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}.$$

Let $\mathbf{Q}$ be a sum-MATLANG query over $\mathcal{S}$ to be evaluated on $\mathcal{I}$. In [GMRV21] the corresponding relational schema is $\{A, R_1^\alpha, R_2^\beta\}$. The instance db is such that $A^{db}(1,1) = 1$, $A^{db}(3,2) = 1$ and 0 for any other tuple. Also, $(R_1^\alpha)^{db}(i) = 1$ for all $i \leq \alpha^{\mathcal{I}} = 3$ and $(R_2^\beta)^{db}(i) = 1$ for all $i \leq \beta^{\mathcal{I}} = 2$. Then, if we translate this relational setting back to a matrix schema and instance, the result is $\mathcal{S}' = \{\mathbf{A}, \mathbf{R}_1, \mathbf{R}_2\}$ with $\mathbf{A}\colon (\alpha, \alpha), \mathbf{R}_1\colon (\alpha, 1)$ and $\mathbf{R}_2\colon (\alpha, 1)$. The instance $\mathcal{I}'$ over $\mathcal{S}'$ is such that $\alpha^{\mathcal{I}} = 3$ (because of the active domain), and:

$$\mathbf{A}^{\mathcal{I}'} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

Additionally, we have $\mathbf{Q}'$ over $\mathcal{S}'$ and that can be evaluated on instance $\mathcal{I}'$. Note that $[\![\mathbf{Q}]\!]_{\mathcal{I}} = [\![\mathbf{Q}']\!]_{\mathcal{I}'}$ does not hold in general, even for the expression $\mathbf{A}$ since $\mathbf{A}^{\mathcal{I}} \neq \mathbf{A}^{\mathcal{I}'}$.

Given the previous discussion, in the following we determine precisely in what sense relations can encode matrices, or matrices can represent relations, and how this correspondence transfers to queries. Then, in Section 4 we show how to generalize the expressibility results in [GMRV21] for any matrix sizes and every encoding between schemas.

3.2. **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 (i.e., $\mathbf{A}$ could be a matrix, a column or row vector, or a scalar), 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 could 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 $\mathbf{A}$ following any of the alternatives above.

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 $\mathbf{A} \simeq R$.

3.3. **How we relate schemas.** A *matrix-to-relational schema encoding* from a matrix schema $\mathcal{S}$ to a relational vocabulary σ is a function $\mathit{Rel} \colon \mathcal{S} \to \sigma$ that maps every matrix symbol $\mathbf{A}$ in $\mathcal{S}$ to a unary or binary relation symbol $\mathit{Rel}(\mathbf{A})$ in σ, and every size symbol α in $\mathcal{S}$ to a constant symbol $\mathit{Rel}(\alpha)$ in σ. Here, $\mathit{Rel}(\mathbf{A})$ 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 $\mathit{Rel}(1) = 1$ 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 $\mathit{Rel}(\mathcal{S})$ for the relational vocabulary σ.

Conversely, we define a *relational-to-matrix schema encoding* from σ into $\mathcal{S}$ as a function $\mathit{Mat} \colon \sigma \to \mathcal{S}$ that maps every relation symbol R to a matrix symbol $\mathit{Mat}(R)$ and every constant symbol c to a size symbol $\mathit{Mat}(c)$. 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 $\mathit{Mat}(\sigma)$ 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.

3.4. **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 3.1. *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.

4. Sum-matlang and positive FO

The previous section has established the formal basis for comparing sum-MATLANG and FO^+. In this section, we give our first expressibility results, proving that sum-MATLANG and FO^+ are equally expressive in some precise sense. Furthermore, we show that its fragments conj-MATLANG and binary conjunctive queries (CQs) are also equally expressive. These results will be our starting point to find fragments of sum-MATLANG that captures fragments of FO^+ with good algorithmic properties, like free-connex and q-hierarchical CQs.

4.1. 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. 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 $\mathit{Rel}(\llbracket \mathbf{Q} \rrbracket(\mathcal{I})) = \llbracket Q \rrbracket_{\mathit{Rel}(\mathcal{I})}$ 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 $\llbracket \mathbf{Q} \rrbracket(\mathcal{I}) = \mathit{Rel}^{-1}(\llbracket Q \rrbracket_{\mathit{Rel}(\mathcal{I})})$, namely, that one can evaluate $\mathbf{Q}$ by first evaluating $\llbracket Q \rrbracket_{\mathit{Rel}(\mathcal{I})}$ and then mapping the results back.

Our main goal will be to prove that one can simulate every sum-MATLANG query in the relational setting by some FO^+ query. For the sake of presentation, we will take a detour by first showing that we can simulate every conj-MATLANG query by some CQ to then leverage this result to sum-MATLANG and FO^+. The first step towards the proof is to note that conj-MATLANG expressions have a prenex normal form as follows. We say that a conj-MATLANG expression $e\colon (\alpha, \beta)$ is in *prenex normal form* if it has the form:

$$e \;\coloneqq\; \Sigma \mathbf{x}, \mathbf{y}, \mathbf{v}_1, \ldots, \mathbf{v}_k.\ \mathbf{s} \times \left(\mathbf{x} \cdot \mathbf{y}^T\right)$$

such that $\mathbf{x}\colon (\alpha, 1)$, $\mathbf{y}\colon (\beta, 1)$, and $\mathbf{s}$ is an expression satisfying the following grammar:

$$\mathbf{s} \coloneqq \mathbf{v}^T \cdot \mathbf{A} \cdot \mathbf{w} \mid \mathbf{v}^T \cdot \mathbf{w} \mid \mathbf{s} \times \mathbf{s}$$

where $\mathbf{A} \in \mathcal{S}$; $\mathbf{v}$ ranges over $\{\mathbf{x}, \mathbf{y}, \mathbf{v}_1, \ldots, \mathbf{v}_k\}$; and $\mathbf{w}$ ranges over $\{\mathbf{x}, \mathbf{y}, \mathbf{v}_1, \ldots, \mathbf{v}_k\}$ and the free vector variables of e.

Lemma 4.1. *Every* conj-MATLANG *expression can be expressed in prenex normal form.*

The proof of Lemma 4.1 is by direct induction on the expression syntax; the interested reader may find it in Appendix B. In essence, the form structure is analogous to the classic conjunctive query structure: the expression $\mathbf{v}^T \cdot \mathbf{A} \cdot \mathbf{w}$ mimics access to base relations using variables $\mathbf{v}$ and $\mathbf{w}$; the expression $\mathbf{v}^T \cdot \mathbf{w}$ mimics equality between $\mathbf{v}$ and $\mathbf{w}$; and $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ are the projected variables. Variables $\mathbf{x}$ and $\mathbf{y}$ encode the size of the output matrix.

By using the prenex normal form of conj-MATLANG, we can prove the simulation of conj-MATLANG queries by CQs.

Proposition 4.2. *For every* conj-MATLANG *query* $\mathbf{Q}$ *over* $\mathcal{S}$ *and every matrix-to-relational schema encoding Rel on* $\mathcal{S}(\mathbf{Q})$*, there exists a CQ* Q *that simulates* $\mathbf{Q}$ *w.r.t. Rel.*

Proof. Let $\mathbf{Q} = \mathbf{H} \coloneqq e$ be a conj-MATLANG query such that $e\colon (\alpha, \beta)$. By definition of queries, e is a sentence (it does not have free variables). Then by Lemma 4.1 we may assume that $e = \Sigma \mathbf{x}, \mathbf{y}, \mathbf{v}_1, \ldots, \mathbf{v}_k.\ \mathbf{s}_1 \times \cdots \times \mathbf{s}_n \times \mathbf{x} \cdot \mathbf{y}^T$ where $\mathbf{s}_i \coloneqq \mathbf{v}^T \cdot \mathbf{A} \cdot \mathbf{w} \mid \mathbf{v}^T \cdot \mathbf{w}$ with $\mathbf{v}, \mathbf{w} \in \{\mathbf{x}, \mathbf{y}, \mathbf{v}_1, \ldots, \mathbf{v}_k\}$.

Let $x, y, v_1, \ldots, v_k$ be FO variables. In what follows, the variables $v, w \in \{x, y, v_1, \ldots, v_k\}$ are selected respectively for $\mathbf{v}, \mathbf{w} \in \{\mathbf{x}, \mathbf{y}, \mathbf{v}_1, \ldots, \mathbf{v}_k\}$: x is used when $\mathbf{x}$ appears, y is used when $\mathbf{y}$ appears, and so on. Additionally, let us denote $\mathit{Rel}(\mathbf{A})$ simply by means of A (in non-bold math font), for every matrix symbol $\mathbf{A}$, and let us denote the constant symbol $\mathit{Rel}(\alpha)$ simply by α, for every size symbol α (recall that $\mathit{Rel}(1) = 1$ by definition).

We now build a CQ Q that simulates $\mathbf{Q}$ w.r.t. *Rel*. For each $1 \leq i \leq n$:

- If $\mathbf{s}_i \coloneqq \mathbf{v}^T \cdot \mathbf{A} \cdot \mathbf{w}$ define:
  - $a_i \coloneqq A(v, w)$ if A is binary.
  - $a_i \coloneqq A(v) \wedge w \leq 1$ if A is unary and $\mathbf{A}\colon (\alpha, 1)$.
  - $a_i \coloneqq v \leq 1 \wedge A(w)$ if A is unary and $\mathbf{A}\colon (1, \alpha)$.
  - $a_i \coloneqq v \leq 1 \wedge w \leq 1 \wedge A()$ if A is nullary.
- If $\mathbf{s}_i \coloneqq \mathbf{v}^T \cdot \mathbf{w}$ with $\mathbf{v}\colon (\gamma, 1)$ and $\mathbf{w}\colon (\gamma, 1)$ define:
  - $a_i \coloneqq v \leq \gamma$ if $\mathbf{v} = \mathbf{w}$.
  - $a_i \coloneqq v \leq \gamma \wedge w \leq \gamma \wedge v = w$ if $\mathbf{v} \neq \mathbf{w}$.

Finally, define $\varphi \coloneqq \exists v_1, \ldots, v_k.a_1 \wedge \cdots \wedge a_n \wedge x \leq \alpha \wedge y \leq \beta$. Note that this formula is in prenex normal form. Then the CQ $Q\colon H(x, y) \leftarrow \varphi$ simulates $\mathbf{Q}$ if $\mathit{Rel}(\mathbf{H})$ is binary and $\mathbf{H}\colon (\alpha, \beta)$; the CQ $Q\colon H(x) \leftarrow \exists y.\varphi$ simulates $\mathbf{Q}$ if $\mathit{Rel}(\mathbf{H})$ is unary and $\mathbf{H}\colon (\alpha, 1)$; the CQ $Q\colon H(y) \leftarrow \exists x.\varphi$ simulates $\mathbf{Q}$ if $\mathit{Rel}(\mathbf{H})$ is unary and $\mathbf{H}\colon (1, \beta)$; and the CQ $Q\colon H() \leftarrow \exists x, y.\varphi$ simulates $\mathbf{Q}$ if $\mathit{Rel}(\mathbf{H})$ is nullary. □

Finally, we can provide a formal proof for the simulation from sum-MATLANG to FO^+.

Proposition 4.3. *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.*

Proof. Let $\mathbf{Q} = \mathbf{H} \coloneqq e$ be a sum-MATLANG query. By definition of query, e is a sentence with $\mathtt{type}(e) = (\alpha, \beta)$ for some α, β. Note that every sum-MATLANG sentence can be expressed as a sum of conj-MATLANG sentences. Indeed, $(e_1 + e_2)^T = e_1^T + e_2^T$; $e_1 \odot (e_1 + e_2) = e_1 \odot e_2 + e_1 \odot e_3$; $(e_1 + e_2) \odot e_3 = e_1 \odot e_3 + e_2 \odot e_3$; $e_1 \cdot (e_2 + e_3) = e_1 \cdot e_2 + e_1 \cdot e_3$; $(e_1 + e_2) \cdot e_3 = e_1 \cdot e_3 + e_2 \cdot e_3$; and $\Sigma \mathbf{v}.\, (e_1 + e_2) = \Sigma \mathbf{v}.e_1 + \Sigma \mathbf{v}.e_2$. Therefore, we can assume that e is of the form $e_1 + \cdots + e_l$ such that each e_i is a conj-MATLANG sentence in prenex normal form.

By Proposition 4.2, for each $i = 1, \ldots, l$ there exists a CQ $Q_i\colon H(\bar{x}_i) \leftarrow \psi_i$ that simulates the query $\mathbf{H} \coloneqq e_i$ w.r.t. *Rel*. Further, since $e = e_1 + \cdots + e_l$ is well-typed, then $e_i\colon (\alpha, \beta)$ for every i. This implies that every formula ψ_i has the same number of free variables and we can assume that $\bar{x}_1 = \ldots = \bar{x}_l = \bar{x}$ without loss of generality. Therefore, the FO^+ query $Q\colon H(\bar{x}) \leftarrow \psi_1 \vee \cdots \vee \psi_l$ simulates $\mathbf{Q}$ where $\psi_1 \vee \cdots \vee \psi_l$ is a safe formula. □

4.2. **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 $\mathit{Mat}(\llbracket Q \rrbracket_{db}) = \llbracket \mathbf{Q} \rrbracket(\mathit{Mat}(db))$ 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 $\llbracket Q \rrbracket_{db} = \mathit{Mat}^{-1}(\llbracket \mathbf{Q} \rrbracket(\mathit{Mat}(db)))$.

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

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

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

$$\frac{\alpha \text{ a size symbol}}{Mat \vdash x = y\colon \{x \mapsto \alpha, y \mapsto \alpha\}}$$

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

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

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

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, a *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 $x \in dom(\tau_1) \cap dom(\tau_2)$. 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 $\texttt{type}(Mat(H)) = (\tau(x_1), \tau(x_2))$; and
- 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 we can show that the query obtained by Proposition 4.3 is always well-typed.

The next proposition connects well-typedness with consistency.

Proposition 4.4. *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 $1 \times \tau(x)^{db}$.*

Proof. We show a more detailed statement. Namely, that for any FO^+ formula φ with $Mat \vdash \varphi\colon \tau$, any database *db* consistent with *Mat* and any variable $x \in free(\varphi)$ if ν is a

valuation with $\nu(x) > \tau(x)^{db}$ then $[\![\varphi]\!]_{db}(\nu) = \mathbb{0}$. The proposition follows from this more general statement since Q is of the form $H(\overline{x}) \leftarrow \varphi$ with $\overline{x}$ a tuple consisting of the free variables of φ (possibly with repetition) and since $\texttt{type}(\mathit{Mat}(H)) = \tau(\overline{x})$ as Q is well-typed under Mat.

The proof of the more detailed statement is by induction on φ. We illustrate the reasoning when φ is an atom, or a disjunction. The other cases are similar.

If φ is a relational atom $R(x_1, x_2)$, then by definition of the typing relation, we know that $(\tau(x_1), \tau(x_2)) = (\alpha, \beta)$ where $\texttt{type}(\mathit{Mat}(R)) = (\alpha, \beta)$. Hence, for valuation ν with $\nu(x_1) > \tau(x_1)^{db}$ or $\nu(x_2) > \tau(x_2)^{db}$ we have $[\![R(x_1, x_2)]\!]_{db}(\nu) = \mathbb{0}$ because R^{db} is consistent with dimensions $\alpha^{db} \times \beta^{db} = \tau(x_1)^{db} \times \tau(x_2)^{db}$. If φ is a comparison atom $x \leq \alpha$, then $\tau(x) = \alpha$ by definition of the typing relation. Hence, for valuation ν with $\nu(x) > \tau(x)^{db} = \alpha^{db}$ we trivially obtain $[\![x \leq \alpha]\!]_{db}(\nu) = \mathbb{0}$.

When φ is a disjunction $\varphi_1 \vee \varphi_2$, let a valuation $\nu\colon \mathit{free}(\varphi)$ such that $\nu(x) > \tau(x)^{db}$ for some $x \in \mathit{free}(\varphi)$. Because we consider only safe formulas, both φ_1 and φ_2 have the same set of free variables, which equals the free variables of φ. Then, from $x \in \mathit{free}(\varphi_1)$ and $x \in \mathit{free}(\varphi_2)$, and so by induction hypothesis both $[\![\varphi_1]\!]_{db}(\nu) = \mathbb{0}$ and $[\![\varphi_2]\!]_{db}(\nu) = \mathbb{0}$ hold. Hence $[\![\varphi]\!]_{db}(\nu) = \mathbb{0}$. □

Similar to the proof from sum-MATLANG to FO^+, we first illustrate how to simulate CQs with conj-MATLANG queries.

Proposition 4.5. *For every binary CQ 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* conj-MATLANG *query $\mathbf{Q}$ that simulates Q w.r.t. Mat.*

Proof. Let $Q\colon H(\overline{x}) \leftarrow \exists v_1, \ldots, v_k.\varphi$ with $\mathit{free}(\varphi) = \{x, y\}$ and $\overline{x} \subseteq \{x, y\}$ be a CQ, i.e., $\varphi = a_1 \wedge \cdots \wedge a_n$ where each a_i is a relational, comparison, or equality atom using variables from the set $\{x, y, v_1, \ldots, v_k\}$. Assume $\mathit{Mat} \vdash Q\colon \tau$.

For simplicity, let all input relations be binary (we will deal with the other cases later). Let $\mathbf{v}, \mathbf{w} \in \{\mathbf{x}, \mathbf{y}, \mathbf{v}_1, \ldots, \mathbf{v}_k\}$ be vector variables of type $(\tau(x), 1), (\tau(y), 1), (\tau(v_1), 1), \ldots, (\tau(v_k), 1)$, respectively. In what follows, we choose $\mathbf{v}, \mathbf{w} \in \{\mathbf{x}, \mathbf{y}, \mathbf{v}_1, \ldots, \mathbf{v}_k\}$ respectively according to $v, w \in \{x, y, v_1, \ldots, v_k\}$. For each $i = 1, \ldots, n$ define:

- $\mathbf{s}_i \coloneqq \mathbf{v}^T \cdot \mathbf{A} \cdot \mathbf{w}$ if a_i is equal to $A(v, w)$. Note that $\mathit{Mat}(A)\colon (\tau(v), \tau(w))$, $\mathbf{v}\colon (\tau(v), 1)$ and $\mathbf{w}\colon (\tau(w), 1)$.
- $\mathbf{s}_i \coloneqq \mathbf{v}^T\mathbf{v}$ if a_i is equal to $v \leq \alpha$. Note that $\mathbf{v}\colon (\tau(v), 1)$.
- $\mathbf{s}_i \coloneqq \mathbf{v}^T\mathbf{w}$ if a_i is equal to $v = w$. Note that $\mathbf{v}\colon (\tau(v), 1)$ and $\mathbf{w}\colon (\tau(w), 1)$, with $\tau(v) = \tau(w)$ because Q is well-typed.

Finally, define

- $e = \Sigma\mathbf{x}, \mathbf{y}, \mathbf{v}_1, \ldots, \mathbf{v}_k.\ (\mathbf{s}_1 \times \cdots \times \mathbf{s}_n) \times \left(\mathbf{x} \cdot \mathbf{y}^T\right)$ if the head of Q is $H(x, y)$.
- $e = \Sigma\mathbf{x}, \mathbf{v}_1, \ldots, \mathbf{v}_k.\ (\mathbf{s}_1 \times \cdots \times \mathbf{s}_n) \times \left(\mathbf{x} \cdot \mathbf{x}^T\right)$ if the head of Q is $H(x, x)$.
- $e = \Sigma\mathbf{x}, \mathbf{v}_1, \ldots, \mathbf{v}_k.\ (\mathbf{s}_1 \times \cdots \times \mathbf{s}_n) \times \mathbf{x}$ if the head of Q is $H(x)$ and $\mathbf{H}\colon (\alpha, 1)$. Otherwise, if $\mathbf{H}\colon (1, \alpha)$, define $e = \Sigma\mathbf{x}, \mathbf{v}_1, \ldots, \mathbf{v}_k.\ (\mathbf{s}_1 \times \cdots \times \mathbf{s}_n) \times \mathbf{x}^T$.

Then $\mathbf{Q} = \mathbf{H} \coloneqq e$ simulates Q w.r.t. Mat for any choice of $\overline{x} \subseteq \{x, y\}$.

If some input is not binary, define:

- $\mathbf{s}_i \coloneqq \mathbf{v}^T \cdot \mathbf{A} \cdot \mathbf{w}$ if $a_i = A(v)$ such that $\mathit{Mat}(A)\colon (\tau(v), 1)$ and $\mathbf{v}\colon (\tau(v), 1)$ for some $\mathbf{w}\colon (1, 1)$.
- $\mathbf{s}_i \coloneqq \mathbf{w}^T \cdot \mathbf{A} \cdot \mathbf{w}$ if $a_i = A()$ such that $\mathit{Mat}(A)\colon (1, 1)$, for some $\mathbf{w}\colon (1, 1)$.

Finally, if there is no $\mathbf{v}_i\colon (1,1)$ in e above we add a dummy $\mathbf{w}\colon (1,1)$ vector variable as $e = \Sigma \mathbf{x}, \mathbf{y}, \mathbf{v}_1, \ldots, \mathbf{v}_k, \mathbf{w}.\ \mathbf{s}_1 \times \cdots \times \mathbf{s}_n \times \mathbf{x} \cdot \mathbf{y}^T$. □

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

Proposition 4.6. *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.*

Proof. First, note that every $\varphi \in \mathsf{FO}^+$ can be written in disjunctive normal form, i.e., $\varphi = \varphi_1 \vee \cdots \vee \varphi_l$ where each φ_i is in prenex normal form and the big disjunction is safe. On one hand, $\exists x.\psi_1 \vee \psi_2 \equiv \exists x.\psi_1 \vee \exists x.\psi_2$ and $(\psi_1 \vee \psi_2) \wedge \psi_3 \equiv (\psi_1 \wedge \psi_2) \vee (\psi_1 \wedge \psi_3)$. On the other hand, if $\psi = R(\overline{x}) \mid x \leq c \mid x = y \mid \psi_1 \wedge \psi_2 \mid \exists x.\psi$ then it can be written in prenex normal form, because $\psi_1 \wedge \exists x.\psi_2 \equiv \exists x.\psi_1 \wedge \psi_2$ since without loss of generality we can assume that x does not appear in φ_1.

Now, let $Q\colon H(\overline{x}) \leftarrow \varphi$ be a binary FO^+ query such that φ is in disjunctive normal form, i.e., $\varphi = \varphi_1 \vee \cdots \vee \varphi_l$. By Proposition 4.5 there exist conj-MATLANG queries $\mathbf{Q}_i = \mathbf{H} \coloneqq e_i$ that simulate the CQs $Q\colon H(\overline{x}) \leftarrow \varphi_i$ w.r.t. *Mat*, respectively.

It is straightforward to see that $\mathbf{Q} = \mathbf{H} \coloneqq e_1 + \cdots + e_l$ simulates Q w.r.t. *Mat*. □

4.3. **Equivalence between Sum-Matlang and positive FO.** Taking into account the correspondence between sum-MATLANG and FO^+ established by Propositions 4.3 and 4.6, 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*.

From Propositions 4.3 and 4.6, and Propositions 4.2 and 4.5 one obtains the following characterization of positive FO and CQs, respectively.

Corollary 4.7. (1) sum-MATLANG *and binary FO^+ queries are equally expressive.*
(2) conj-MATLANG *and binary conjunctive queries are equally expressive.*

While the result between conj-MATLANG and CQs 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.

5. Free-connex conjunctive queries and $\mathsf{FO}^\wedge_2$

We wish to understand the fragment of MATLANG that allows efficient enumeration-based evaluation, i.e., enumerating the query result with constant delay after a preprocessing phase that is linear in the input instance. Towards that goal, we will first specialize in this section and the next the correspondence between conj-MATLANG and CQs given by Corollary 4.7 to *free-connex* CQs [BDG07, Bra13]. Next, we recall the definition of free-connex CQs and state the main result of this section.

5.1. **Free-connex CQs.** First, recall that a query Q is a CQ if it is in prenex normal form:

$$Q : H(\overline{x}) \leftarrow \exists \overline{y}.a_1 \wedge \cdots \wedge a_n \qquad (*)$$

where $a_1, \ldots, a_n$ are relational, comparison, or equality atoms. From now on, unless explicitly stated otherwise, we will assume without loss of generality that a CQ does not have equality atoms. Indeed, we can always remove equality atoms from CQ by taking the equivalence classes formed by all equalities in the body and replacing each variable in the body and head of the query with a representative from its class.

A CQ Q is *acyclic* (also sometimes referred to as α-acyclic [Fag83]) if it has a *join tree*. A join tree for Q is an undirected tree T whose node-set is the set of all atoms (i.e., relational or comparison atoms) occurring in Q, that satisfies the *connectedness* property. This property requires that for every variable x occurring (free or bound) in Q, all the nodes containing x form a connected subtree of T. Note that we consider comparison atoms as unary. Then, a CQ Q like (*) is *free-connex* 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 requirement that the query Q' obtained by adding the head atom $H(\overline{x})$ to the body of Q is also acyclic forbids queries like $H(x, y) \leftarrow \exists z.A(x, z) \wedge B(z, y)$. Indeed: 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.

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 [BDG07].

In this section, we prove the following correspondence between fc-CQs and $\mathsf{FO}^{\wedge}_2$, the two-variable fragment of $\mathsf{FO}^{\wedge}$. Formally, a formula φ in $\mathsf{FO}^{\wedge}$ is said to be in $\mathsf{FO}^{\wedge}_2$ if it does not use equality and 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 5.1. *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, we will use this result in Section 6 to identify the fragment of MATLANG queries that expressively correspond to fc-CQs. Having that fragment in hand, we later turn to its actual query evaluation on arbitrary (not necessarily Boolean) semirings in Section 10.

The rest of this section is devoted to proving Theorem 5.1. We do so in three steps. First, in Section 5.2 we define *query plans*, which allow to give an alternate but equivalent definition of fc-CQs that we find more convenient to use. Then, in Sections 5.3 and 5.4 we prove the two directions of Theorem 5.1.

5.2. **Query plans.** To simplify notation, in what follows we denote the set of all FO variables that occur in an object X by $var(X)$. In particular, if X is itself a set of variables, then $var(X) = X$.

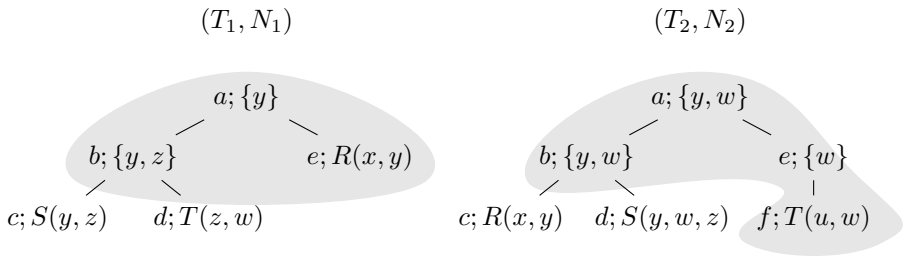


FIGURE 2. Two query plans (T_1, N_1) and (T_2, N_2) where the sets N_1 and N_2 are indicated by the gray area.

A *query plan* (QP for short) is a pair (T, N) with T a *generalized join tree* and N a *sibling-closed connex subset of* T. A generalized join tree (GJT) is a node-labeled directed tree $T = (V, E)$ such that

- T is binary: every node has at most two children.
- Every leaf is labeled by an atom (relational or inequality).
- Every interior node n is labeled by a set of variables and has at least one child c such that $var(n) \subseteq var(c)$. Such child c is called the *guard* of n.
- Connectedness property: for every variable x the sub-graph consisting of the nodes n where $x \in var(n)$ is connected.

A *connex subset of* T is a set $N \subseteq V$ that includes the root of T such that the subgraph of T induced by N is a tree. N is *sibling-closed* if for every node $n \in N$ with a sibling m in T, also m is in N. The *frontier* of a connex set N is the subset $F \subseteq N$ consisting of those nodes in N that are the leaves in the subtree of T induced by N.

Example 5.2. To illustrate, Figure 2 shows two query plans. There and in future illustrations we use the convention that we depict a node with id n and label l as $n; l$. The elements of the query plan's connex set are indicated by the gray area. So, in T_1, we have $N_1 = \{a, b, e\}$ and in T_2 we have $N_2 = \{a, b, e, f\}$. Since all nodes in a connex set must be connected, an unallowed choice of N_1 for the tree T_1 would be $\{a, c\}$. Furthermore, also $\{a, b\}$ is not allowed, since the set must be sibling-closed.

For ease of presentation, we abbreviate query plans of the form $(T, \{r\})$ simply by (T, r) when the connex set consists only of the single root node r. Furthermore, if T is a GJT and n is a node of T, then we write $T|_n$ for the subtree of T rooted at n.

Let (T, N) be a query plan and assume that $\{\!\{a_1, \ldots a_n\}\!\}$ is the multiset of atoms occurring as labels in the leaves of T. Then the *formula associated to* (T, N) is the conjunctive formula

$$\varphi[T, N] \;:=\; \exists y_1 \ldots \exists y_k.\, a_1 \wedge \ldots \wedge a_n$$

where $var(T) \setminus var(N) = \{y_1, \ldots, y_k\}$. In particular, $var(N) = free(\varphi[T, N])$. A query plan (T, N) is a *query plan for* a CQ Q with body φ if $\varphi[T, N] \equiv \varphi$.

Example 5.3. Referring to Figure 2, we have

$$\varphi[T_1, N_1] = \exists w.\, S(y, z) \wedge T(z, w) \wedge R(x, y),$$
$$\varphi[T_2, N_2] = \exists x \exists z.\, R(x, y) \wedge S(y, w, z) \wedge T(u, w).$$

The following proposition is a particular case of the results shown in [IUV17, IUV$^+$20].

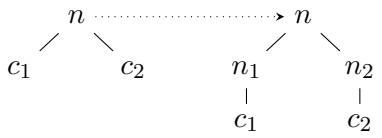
Proposition 5.4. *A CQ Q is free-connex if, and only if, it has a query plan.*

5.3. **From binary free-connex CQ to the two variable fragment of FO.** We next show that it is possible to translate every binary fc-CQ Q into an equivalent $\mathsf{FO}_2^\wedge$ query. Our construction is by induction on a suitable query plan for Q. To enable this induction, we first observe the following properties of query plans.

Two query plans (T, N) and (T', N') are said to be *equivalent* if (1) the multiset of atoms $at(T)$ appearing in T equals the multiset of atoms $at(T')$ appearing in T', and (2) $var(N) = var(N')$.[1] Clearly if two query plans are equivalent then their associated formulas are also equivalent.

Lemma 5.5. *For every QP there exists an equivalent QP such that for any node n with two children c_1, c_2 it holds that $var(c_1) \subseteq var(n)$ and $var(c_2) \subseteq var(n)$. In particular, either $var(c_1) = var(n)$ or $var(c_2) = var(n)$ holds.*

Proof. Let n be any node of T that has two children c_1 and c_2. Let (T, N) be the original query plan. We modify (T, N) into an equivalent plan (T', N') that satisfies the lemma by adding, for every node n with two children c_1 and c_2, two new intermediate nodes n_1 and n_2 as follows labeled with $var(n_1) = var(n) \cap var(c_1)$ and $var(n_2) = var(n) \cap var(c_2)$.



We obtain the set N' by adding also n_i to N if $c_i \in N$, $i = 1, 2$. It is readily verified that T' is a GJT; N' is a sibling-closed connex subset of T'; that T and T' have the same multiset of atoms; and that $var(N) = var(N')$. Hence $(T, N) \equiv (T', N')$. Moreover, in T' every node n with two children n_1 and n_2 is such that $var(n_1) \subseteq var(n)$ and $var(n_2) \subseteq var(n)$. Finally, because T is a GJT, it is the case that either $var(n) \subseteq var(c_1)$ or $var(n) \subseteq var(c_2)$. As such, either $var(n) = var(n_1)$ or $var(n) = var(n_2)$. □

Next, we show how the formulas and subformulas encoded by a query plan are related.

Lemma 5.6. *Let (T, M) and (T, N) be two QPs over the same GJT such that $N \subseteq M$. Then $\varphi[T, N] \equiv \exists \overline{z}.\varphi[T, M]$ with $\overline{z} = var(M) \setminus var(N)$.*

Proof. Let $\{\!\{a_1, \ldots, a_k\}\!\}$ be the multiset of atoms in T. Let $\overline{x}$ be the set of all variables occurring in $a_1, \ldots, a_k$ that are not mentioned in $var(N)$. Let $\overline{y}$ be the set of all variables occurring in $a_1, \ldots, a_k$ that are not mentioned in $var(M)$. Because $N \subseteq M$ also $var(N) \subseteq var(M)$ and hence $\overline{x} \supseteq \overline{y}$. Moreover, $\overline{x} = \overline{y} \cup \overline{z}$. Then, by definition,

$$\varphi[T, N] = \exists \overline{x}.(a_1 \wedge \cdots \wedge a_k) \equiv \exists \overline{z}.\exists \overline{y}.(a_1 \wedge \cdots \wedge a_k) \equiv \exists \overline{z}.\varphi[T, M]$$ □

Lemma 5.7. *Let (T, N) be a query plan and let $F = \{r_1, \ldots, r_l\}$ be the frontier of N, i.e, the bottom nodes of N. Then $\varphi[T, N] \equiv \varphi[T|_{r_1}, r_1] \wedge \cdots \wedge \varphi[T|_{r_l}, r_l]$.*

[1] Note that in condition (1) we view $at(T)$ and $at(T')$ as *multisets*: so T and T' must have the same set of atoms, and each atom must appear the same number of times in both.

Proof. Let, for every frontier node r_i, A_i be the multiset of atoms appearing in $T|_{r_i}$. Then, by definition

$$\varphi[T,N] = \exists \overline{u}.\ \bigwedge_{a_1 \in A_1} a_1 \wedge \cdots \wedge \bigwedge_{a_l \in A_l} a_l$$

where $\overline{u}$ is the sequence of all variables in T except the variables in N.

We can write $\overline{u}$ as union of l sets $\overline{v_1} \cup \cdots \cup \overline{v_l}$ where $\overline{v_i}$ consists of all variables in T_i except $var(N)$. Because T is a GJT, it satisfies the connectedness property, and hence we know that any variable that is shared by atoms in T_i and atoms in T_j, for $1 \leq i < j \leq l$ must be in $var(r_i) \cap var(r_j) \subseteq var(N)$. It follows that the sets $\overline{v_i}$ are disjoint: assume for the purpose of obtaining a contradiction that variable x occur in both $\overline{v_i}$ and $\overline{v_j}$. Then it occurs in T_i and T_j, and hence, by the running intersection property of T in $var(r_i) \cap var(r_j)$. Therefore, $x \in var(N)$, which gives the desired contradiction. Then we can rewrite $\varphi[T,N]$ as follows

$$\begin{aligned}
\varphi[T,N] &= \exists \overline{u}.\ \bigwedge_{a_1 \in A_1} a_1 \wedge \cdots \wedge \bigwedge_{a_l \in A_l} a_l \\
&= \exists \overline{v_1} \exists \overline{v_2} \ldots \exists \overline{v_l}.\ \bigwedge_{a_1 \in A_1} a_1 \wedge \cdots \wedge \bigwedge_{a_l \in A_l} a_l \\
&\equiv \left(\exists \overline{v_1}.\ \bigwedge_{a_1 \in A_1} a_1 \right) \wedge \cdots \wedge \left(\exists \overline{v_l}.\ \bigwedge_{a_l \in A_l} a_l \right) \\
&\equiv \varphi[T_1, r_1] \wedge \cdots \wedge \varphi[T_l, r_l].
\end{aligned}$$

Here, the last equivalence holds because $\overline{v_i}$ is exactly the set of variables in T_i except $var(r_i)$. This can be seen as follows. By definition, $\overline{v_i} = var(T_i) \setminus var(N)$. Because $var(N) \supseteq var(r_i)$, it follows that $\overline{v_i} \subseteq var(T_i) \setminus var(r_i)$. Now assume, for the purpose of obtaining a contradiction that this inclusion is strict, i.e., that there is some $x \in var(T_i) \cap var(N)$ that is not in $var(r_i)$. Then, because it is in $var(N)$ there is some $r_j \neq r_i$ with $x \in var(r_j)$. Because also $x \in var(T_i)$, it appears in an atom of T_i and is shared with an atom of T_j. Because T has the running intersection property, we know that hence $x \in var(r_i) \cap var(r_j)$, which is the desired contradiction since then $x \in var(r_i)$. □

Corollary 5.8. *Let (T,r) be a query plan where r has one child c. Then $\varphi[T,r] \equiv \exists \overline{y}.\varphi[T|_c, c]$ with $\overline{y} = var(c) \setminus var(r)$.*

Proof. Consider the set $M = \{r, c\}$, which is connex in T. By Lemma 5.6 we have $\varphi[T,r] \equiv \exists \overline{y}.\varphi[T,M]$. Moreover, since the frontier of M is $\{c\}$ we obtain by Lemma 5.7 that $\varphi[T,M] \equiv \varphi[T|_c, c]$. Hence $\varphi[T,r] \equiv \exists \overline{y}.\varphi[T|_c, c]$. □

Corollary 5.9. *Let (T,r) be a query plan where r has two children r_1, r_2 such that one of the following holds:*

- *$var(r_1) = var(r)$ and $var(r_2) \subseteq var(r)$; or*
- *$var(r_2) = var(r)$ and $var(r_1) \subseteq var(r)$.*

Then $\varphi[T,r] \equiv \varphi[T|_{r_1}, r_1] \wedge \varphi[T|_{r_2}, r_2]$.

Proof. Let $N = \{r, r_1, r_2\}$. This set is a connex subset of T. Hence, (T,N) is also a query plan. Observe that, because $var(N) = var(r)$ by the assumptions on $var(r_1)$ and $var(r_2)$, we have $\varphi[T,r] = \varphi[T,N]$. Further observe that the frontier of N is $\{r_1, r_2\}$. Then, by Lemma 5.7 $\varphi[T,r] = \varphi[T,N] \equiv \varphi[T_1, r_1] \wedge \varphi[T_2, r_2]$ as desired. □

Finally, we note a property of query plans over binary relations.

Lemma 5.10. *Let (T, N) be a query plan such that $\varphi[T, N]$ is binary then $|\, var(N)| \leq 2$ and any node of T contains at most two variables.*

Proof. By definition $var(N) = free(\varphi[T, N])$, then necessarily $|\, var(N)| \leq 2$ because $\varphi[T, N]$ is binary. Now, suppose that there exists some node g of T that has more than two variables. This implies that the child which is the guard of g also has more than two variables, and so on. Thus there is an leaf atom that uses more than two variables, which is not possible since $\varphi[T, N]$ is binary. □

At this point we are ready to prove that we can translate any binary query plan of the form (T, r) into an equivalent $\mathsf{FO}_2^\wedge$ formula.

Proposition 5.11. *For every QP (T, r) such that $\varphi[T, r]$ is binary, there exists an $\mathsf{FO}_2^\wedge$ formula ψ equivalent to $\varphi[T, r]$.*

Proof. By Lemma 5.5 we may assume without loss of generality that for every node n in T with children c_1, c_2 either $var(c_1) = var(n)$ and $var(c_2) \subseteq var(n)$; or $var(c_2) = var(n)$ and $var(c_1) \subseteq var(n)$. Moreover, by Lemma 5.10 we know that all nodes n in T have $|\, var(n)| \leq 2$. We build the $\mathsf{FO}_2^\wedge$ formula ψ equivalent to $\varphi[T, r]$ by induction on the height of T.

The base case is where the height of T is zero. Then T consists of just its root, r, which must be labeled by atom a and $\varphi[T, r] = a$. Then it suffices to take $\psi := a$. Clearly, $\psi \equiv \varphi[T, r]$ and, because a is of arity at most two by assumption, $\psi \in \mathsf{FO}_2^\wedge$.

Now, for the inductive step. Assume T has height $k > 0$. By inductive hypothesis we have that the statement holds for height $k - 1$. We distinguish the cases where r has only one child or it has two children.

- First, if r has only one child c. By induction hypothesis there exists a formula $\psi_c \in \mathsf{FO}_2^\wedge$ such that $\psi_c \equiv \varphi[T|_c, c]$. Note that this implies that $free(\psi_c) = free(\varphi[T|_c, c])$. Take $\psi := \exists\, var(c) \setminus var(r).\psi_c$. By Corollary 5.8 we have
$$\psi = \exists\, var(c) \setminus var(r).\psi_c \equiv \exists\, var(c) \setminus var(r).\varphi[T|_c, c] \equiv \varphi[T, r].$$
Clearly, $\psi \in \mathsf{FO}_2^\wedge$, since it is a projection of $\psi_c \in \mathsf{FO}_2^\wedge$.
- Second, assume that r has two children, r_1 and r_2. We may assume w.l.o.g. that $var(r_1) = var(r)$ and $var(r_2) \subseteq var(r)$. By the inductive hypothesis there exist $\mathsf{FO}_2^\wedge$ formulas $\psi_1 \equiv \varphi[T|_{r_1}, r_1]$ and $\psi_2 \equiv \varphi[T|_{r_2}, r_2]$. We next construct an $\mathsf{FO}_2^\wedge$ formula equivalent to $\psi_1 \wedge \psi_2$. This suffices since by Corollary 5.9 $\varphi[T|_{r_1}, r_1] \wedge \varphi[T|_{r_2}, r_2] \equiv \varphi[T, r]$ and hence $\psi_1 \wedge \psi_2 \equiv \varphi[T, r]$.

  Note that $free(\psi_1) = var(r_1) = var(r)$ and $free(\psi_2) = var(r_2) \subseteq var(r)$. Moreover, $\varphi[T, r]$ is binary. Thus, there exists a set of two variables, say $\{x, y\}$, such that $var(r) \subseteq \{x, y\}$. Consequently, $free(\psi_1) \subseteq \{x, y\}$, $free(\psi_2) \subseteq \{x, y\}$. Because $\psi_1, \psi_2 \in \mathsf{FO}_2^\wedge$ we know that when ψ_1 (or ψ_2) has exactly two free variables, x and y are the only variables used in ψ_1 (resp. ψ_2). When ψ_1 (resp. ψ_2) has fewer than two free variables it is possible that also some other variable $z \notin \{x, y\}$ occurs in ψ_1. However, because $\psi_1 \in \mathsf{FO}_2^\wedge$ it is then always possible to rename bound variables to use only x, y. The same holds for ψ_2. We conclude that there exist formulas $\psi_1', \psi_2' \in \mathsf{FO}_2^\wedge$ such that $\psi_1 \equiv \psi_1'$, $\psi_2 \equiv \psi_2'$ and ψ_1', ψ_2' only use the variables x and y. Thus $\varphi[T|_{r_1}, r_1] \wedge \varphi[T|_{r_2}, r_2] \equiv \psi_1' \wedge \psi_2' \in \mathsf{FO}_2^\wedge$. □

Finally, we are able to show that every formula encoded by a binary query plan has an equivalent $\mathsf{FO}_2^\wedge$ formula.

Proposition 5.12. *For every QP (T, N) such that $\varphi[T, N]$ is binary, there exists an $\mathsf{FO}_2^\wedge$ formula ψ equivalent to $\varphi[T, N]$.*

Proof. By Lemma 5.5 we may assume w.l.o.g. that for every node n in T with two children c_1 and c_2 we have $var(c_1) = var(n)$ and $var(c_2) \subseteq var(n)$; or $var(c_1) = var(n)$ and $var(c_2) \subseteq var(n)$. Assume that the frontier of N is $F = \{r_1, \ldots, r_l\}$. By Proposition 5.11, there are $\mathsf{FO}_2^\wedge$ formulas $\psi_1, \ldots, \psi_l$ equivalent to $\varphi[T|_{r_1}, r_1], \ldots, \varphi[T|_{r_l}, r_l]$, respectively. By Lemma 5.7 we have $\varphi[T, N] \equiv \varphi[T|_{r_1}, r_1] \wedge \cdots \wedge \varphi[T|_{r_l}, r_l] \equiv \psi_1 \wedge \cdots \wedge \psi_l$. To prove the proposition, it hence suffices to show that the conjunction $\psi_1 \wedge \ldots \wedge \psi_l$ is expressible in $\mathsf{FO}_2^\wedge$. For this we reason as follows.

By assumption $\varphi[T, N]$ is binary, thus there exists a set of two variables, say $\{x, y\}$, such that $var(N) \subseteq \{x, y\}$. Hence, $var(r_i) \subseteq var(N) \subseteq \{x, y\}$. Then, because ψ_i is equivalent to $\varphi[T_i, r_i]$, also $free(\psi_i) = var(r_i) \subseteq \{x, y\}$ for $i = 1, \ldots, l$. Similar to our argumentation in the proof of Proposition 5.11 we can then exploit the fact that every $\psi_i \in \mathsf{FO}_2^\wedge$ to obtain formulas $\psi'_1, \ldots, \psi'_l \in \mathsf{FO}_2^\wedge$ such that $\psi'_i \equiv \psi_i$ and ψ'_i only uses the variables x and y, for $i = 1, \ldots, l$. The formulas ψ'_i are obtained from ψ_i by renaming bound variables, which is only necessary if ψ_i has fewer than two free variables but also mentions some bound variable $z \notin \{x, y\}$. Thus $\varphi[T, N] \equiv \varphi[T|_{r_1}, r_1] \wedge \cdots \wedge \varphi[T|_{r_l}, r_l] \equiv \psi'_1 \wedge \ldots \wedge \psi'_l \in \mathsf{FO}_2^\wedge$ □

5.4. **From the two variable fragment of FO to binary free-connex CQ.** We end this section by proving the other direction, namely, that every formula in $\mathsf{FO}_2^\wedge$ is equivalent to some binary free-connex CQ. Similar to the previous subsection, we use binary query plans.

Proposition 5.13. *For every $\mathsf{FO}_2^\wedge$ formula ψ there exists a binary QP (T, N) such that $\varphi[T, N]$ is equivalent to ψ.*

Proof. Fix two distinct variables x and y. Because $\mathsf{FO}_2^\wedge$ formulas use at most two variables, we may assume without loss of generality that the only variables occurring in $\mathsf{FO}_2^\wedge$ formulas are x and y. We obtain the proposition by proving a more detailed statement: for every $\psi \in \mathsf{FO}_2^\wedge$ there exists a binary QP (T, N) such that $\varphi[T, N] \equiv \psi$ and either:

- $N = \{r\}$ with r the root node r; or
- N consists of three nodes, $N = \{r, c_1, c_2\}$ with r the root of T and c_1, c_2 its children, which are labeled by $var(r) = \emptyset$, $var(c_1) = \{x\}$, and $var(c_2) = \{y\}$, respectively. This case only happens when ψ is a cross product $\psi = \psi_1 \wedge \psi_2$.

Note that in the second case it follows from Lemma 5.7 that $\psi = \psi_1 \wedge \psi_2 \equiv \varphi[T, N] \equiv \varphi[T|_{c_1}, c_1] \wedge \varphi[T|_{c_2}, c_2]$ since the frontier of $\{r, c_1, c_2\}$ is $\{c_1, c_2\}$.

We prove the stronger statement by induction on ψ.

- When ψ is an atom then the result trivially follows by creating GJT T with a single node r, labeled by the atom, and fixing $N = \{r\}$.
- When $\psi = \exists z.\psi'$ with $z \in \{x, y\}$ we may assume w.l.o.g. that $z \in free(\psi')$: if not then $\psi \equiv \psi'$ and the result follows directly from the induction hypothesis on ψ'. So, assume $z \in free(\psi')$. By induction hypothesis we have a binary QP (T', N') equivalent to ψ'. We next consider two cases.
  - N' is a singleton, $N' = \{r'\}$ with r' the root of T'. We create the claimed binary QP (T, N) as follows: let T be obtained by adjoining a new root r to T'; label r by $var(r') \setminus \{z\}$; and make r' the only child of r. Take $N = \{r\}$. Then (T, N) remains binary. Equivalence of $\varphi[T, N]$ to ψ follows by induction hypothesis and Lemma 5.8.

– Otherwise $N' = \{r', c_1, c_2\}$ with c_1 and c_2 the children of r' in T'; $var(r') = \emptyset$; $var(c_1) = \{x\}$; $var(c_2) = \{y\}$; and $\psi' \equiv \varphi[T'|_{c_1}, c_1] \wedge \varphi[T'|_{c_2}, c_2]$. Since $z \in \{x, y\}$ we have either $var(c_1) = \{z\}$ or $var(c_2) = \{z\}$ (but not both). Assume w.l.o.g. that $var(c_1) = \{z\}$; the other case is similar. We create the claimed QP (T, N) as follows: let T be obtained by removing r' from T' and replacing it with a new root, r, which has children c_1 and c_2 and which is labeled by $var(c_2)$. Since $var(c_1)$ and $var(c_2)$ were disjoint, this new tree satisfies the guardedness and connectedness properties, and remains binary. Take $N = \{r\}$. Then

$$var(N) = var(c_2) = var(N') \setminus var(c_1) = free(\psi') \setminus \{z\} = free(\psi),$$

as desired. To see why $\varphi[T, N] \equiv \psi$ first define $M = \{r, c_1, c_2\}$. By lemma 5.6 we obtain that $\varphi[T, N] \equiv \exists z.\varphi[T, M]$. Furthermore, since the frontier of M is $\{c_1, c_2\}$ we know from Lemma 5.7 that $\varphi[T, M] \equiv \varphi[T|_{c_1}, c_1] \wedge \varphi[T|_{c_2}, c_2] = \varphi[T'|_{c_1}, c_1] \wedge \varphi[T'|_{c_2}, c_2] \equiv \psi'$. Hence $\varphi[T, N] \equiv \exists z.\psi' = \psi$.

- Otherwise ψ is a conjunction, $\psi = \psi_1 \wedge \psi_2$. By induction hypothesis, we have binary QPs (T_1, N_1) and (T_2, N_2) equivalent to ψ_1 and ψ_2, respectively. We may assume w.l.o.g. that T_1 and T_2 have no nodes in common, as we can always rename nodes otherwise. We may further assume w.l.o.g. that $var(T_1) \cap var(T_2) \subseteq var(N_1) \cap var(N_2)$, i.e., that all variables that are mentioned in both T_1 and T_2 are also in $var(N_1) \cap var(N_2)$. Indeed, for every variable $z \in (var(T_1) \cap var(T_2)) \setminus (var(N_1) \cap var(N_2))$ we can simply rename all occurrences of z in T_1 to a new unique variable z' that does not occur anywhere in T_2. Assume that we apply this reasoning to all such variables Z and let T_1' be the result of applying this renaming on T_1. Note that N_1 is still a connex subset of T_1'. Moreover, $\varphi[T_1, N_1] \equiv \varphi[T_1', N_1]$: essentially we have only renamed bound variables in the formula $\varphi[T_1, N_1]$ to obtain $\varphi[T_1', N_1]$. So we can continue our reasoning with (T_1', N_1) instead of (T_1, N_1) and the former has no variables in common with T_2, except those in $var(N_1) \cap var(N_2)$.

  By our inductive hypothesis, N_1 and N_2 are either singletons, or contain exactly three nodes. We consider three cases.

  – Both N_1 and N_2 are singletons, say $N_1 = \{r_1\}$ and $N_2 = \{r_2\}$ with r_1, r_2 the roots of T_1, T_2 respectively. In particular, because $\varphi[T_i, N_i]$ is equivalent to ψ_i for $i = 1, 2$ we necessarily have $var(r_i) = free(\psi_i)$. Construct the claimed binary QP (T, N) by taking the (disjoint) union of T_1 and T_2 and adjoining a new root node r with children r_1, r_2 such that $var(r) = var(r_1) \cap var(r_2)$. Since the only variables shared between T_1 and T_2 are in $var(N_1) \cap var(N_2) = var(r_1) \cap var(r_2)$ it follows that T satisfies the connectedness property, and is hence a GJT. Moreover, let $N = \{r, r_1, r_2\}$. Clearly

$$var(N) = var(r_1) \cup var(r_2) = free(\psi_1) \cup free(\psi_2) = free(\psi) \subseteq \{x, y\}.$$

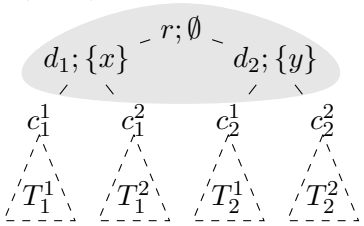
    Correctness follows by Lemma 5.7, since r_1 and r_2 are the frontier nodes of N:

$$\varphi[T, N] \equiv \varphi[T|_{r_1}, r_1] \wedge \varphi[T|_{r_2}, r_2] = \varphi[T_1, r_1] \wedge \varphi[T_2, r_2] \equiv \psi_1 \wedge \psi_2.$$

  – Both N_1 and N_2 contain exactly three nodes, say $N_1 = \{r_1, c_1^1, c_2^1\}$ and $N_2 = \{r_2, c_1^2, c_2^2\}$ with r_i the root node of T_i and c_j^i the two children of r_i in T_i, for $i, j \in \{1, 2\}$. Furthermore, $var(r^i) = \emptyset$; $var(c_1^i) = \{x\}$ and $var(c_2^i) = \{y\}$ for $i = 1, 2$ and ψ_1 and ψ_2 must themselves be conjunctions, $\psi_1 = \psi_1^1 \wedge \psi_2^1$ and $\psi_2 = \psi_1^2 \wedge \psi_2^2$. By induction hypothesis,

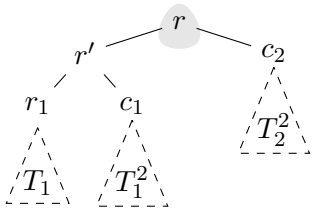
$$\psi_i \equiv \varphi[T_i, N_i] \equiv \varphi[T_i|_{c_1^i}, c_1^i] \wedge \varphi[T_i|_{c_2^i}, c_2^i] \tag{5.1}$$

We construct the claimed QP (T, N) by taking T as follows, and $N = \{r, d_1, d_2\}$.



Here r, d_1 and d_2 are new nodes with $var(r) = \emptyset$; $var(d_1) = \{x\}$, and $var(d_2) = \{y\}$, and $T_j^i = T_i|_{c_j^i}$ for $i = 1, 2$. It is straightforward to verify that T satisfies the guardedness and connectedness properties. It is hence a GJT. To see why $\varphi[T, N] \equiv \psi$, first define $M = N \cup \{c_1^1, c_2^1, c_1^2, c_2^2\}$. Note that $var(M) = var(N)$. Hence, by Lemma 5.6, we have $\varphi[T, N] \equiv \varphi[T, M]$. Then, because the frontier of M is $\{c_1^1, c_2^1, c_1^2, c_2^2\}$ we have by Lemma 5.7 that $\varphi[T, M] \equiv \bigwedge_{i,j\in\{1,2\}} \varphi[T|_{c_j^i}, c_j^i] \equiv \bigwedge_{i,j\in\{1,2\}} \varphi[T_i|_{c_j^i}, c_j^i] \equiv \psi_1 \wedge \psi_2$, where the last equivalence follows from (5.1).

– One of N_1 and N_2 has exactly three nodes; the other has one node. Assume w.lo.g. that $N_1 = \{r_1\}$ and $N_2 = \{r_2, c_1, c_2\}$. We then construct the claimed binary QP (T, N) as follows. If $var(r_1) = \{x, y\}$ then we take T as follows, where r, r' are new nodes with $var(r) = var(r') = var(r_1) = \{x, y\}$ and $N = \{r\}$. Moreover, $T_j^2 = T_2|_{c_j}$ for $j = 1, 2$.



When $var(r_1) \neq \{x, y\}$ then either $var(r_1) \subseteq var(c_1)$ or $var(r_1) \subseteq var(c_2)$ (or both). If $var(r_1) \subseteq var(c_1)$ then we take T as above but set $var(r) = \emptyset$; $var(r') = var(c_1) = \{x\}$; and $N = \{r, r_1, c_2\}$. The case where $var(r_1) \subseteq var(c_2^2)$ but $var(r_1) \not\subseteq var(c_1)$ is similar. In all cases, equivalence with ψ follows from the same reasoning used earlier. □

6. The free-connex fragment of matlang queries

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.

In this section, we prove the following correspondence.

Theorem 6.1. fc-MATLANG *and* $\mathsf{FO}_2^\wedge$ *are equally expressive.*

Combined with Theorem 5.1 we obtain the correspondence with binary free-connex CQ:

Corollary 6.2. fc-MATLANG *and binary free-connex CQs are equally expressive.*

We prove both directions of Theorem 6.1 separately.

6.1. **From $\mathsf{FO}_2^\wedge$ to fc-MATLANG.** We first simulate $\mathsf{FO}_2^\wedge$ queries by means of fc-MATLANG queries. In order to achieve this, free variables of formulas need to describe row and column indexes of expressions. Moreover, it is crucial to know which free variable will correspond to the row index and which free variable will correspond to the column index. For this reason we assume the existence of an arbitary *canonical order* $\prec$ over FO variables, where $x \prec y$ denotes that x strictly precedes y in this order.

Fix a vocabulary σ. We first define the notion of when a sum-MATLANG expression simulates a *formula* over σ instead of a query. Let $\psi \in \mathsf{FO}^+$ over σ. Let *Mat* be a relational-to-matrix schema encoding on σ such that $Mat \vdash \psi \colon \tau$. We say that sum-MATLANG sentence e *simulates* ψ *w.r.t. Mat* if the sum-MATLANG query $Mat(ans) := e$ simulates the query $ans(\overline{x}) \leftarrow \psi$ w.r.t. *Mat*, where

(1) $\overline{x}$ is a tuple containing exactly the free variables of ψ, without repetition and canonically ordered: i.e., if $\overline{x} = (x_1, x_2)$ then $x_1 \prec x_2$.
(2) *ans* is an arbitrary relation symbol, not occurring in φ, of the same arity as $free(\varphi)$ such that $\mathtt{type}(Mat(ans)) = \tau(\overline{x})$.

So, in the second bullet if φ has two free variables then $\mathtt{type}(Mat(ans)) = (\tau(x_1), \tau(x_2))$; if φ has one free variable then $\mathtt{type}(Mat(ans)) = (\tau(x), 1)$; and if φ has zero free variables then $\mathtt{type}(Mat(ans)) = (1, 1)$. In particular, due to well-typedness, when φ has two free variables $Mat(ans)$ is consistent with dimensions $\tau(x_1)^{db} \times \tau(x_2)^{db}$; with $\tau(x)^{db} \times 1$ when φ has one free variable; and with 1×1 when φ has no free variable, for every database db consistent with *Mat*. Further, we note that, while relation symbol *ans* is fixed arbitrarily, it is clear that if e simulates φ w.r.t. one particular choice of *ans*, then it simulates it w.r.t. all valid choices.

It is important to stress that (1) in the query $ans(\overline{x}) \leftarrow \varphi$ no variable is ever repeated in the head; and (2) if φ hence has a single free variable, and hence computes a unary relation, then the simulating fc-MATLANG expression will always simulate it by means of a *column* vector. We fix this merely to simplify the proof, and could also have fixed it to be a row vector instead.

Proposition 6.3. *Let $\psi \in \mathsf{FO}_2^\wedge$ over σ. Let Mat be a relational-to-matrix schema encoding on σ such that $Mat \vdash \psi \colon \tau$. Then there exists an expression $e \in$ fc-MATLANG that simulates ψ w.r.t. Mat.*

Proof. Fix two distinct variables $x_1 \prec x_2$. We may assume without loss of generality that the only variables occurring in ψ are x_1, x_2. We prove the statement by induction on ψ. For every relation symbol R, we denote $Mat(R)$ simply by $\mathbf{R}$. In particular, $\mathbf{ans} = Mat(ans)$.

First, when $\psi = a$ and a is a relational atom we have multiple cases.

- Atom a is a binary atom, and its variables occur in canonical order, i.e. $a = R(x_1, x_2)$. Then take $e := \mathbf{R}$.
- Atom a is a binary atom, but its variables occur in reverse canonical order, i.e., $a = R(x_2, x_1)$. Then take $e := \mathbf{R}^T$.
- Atom a is a binary atom that mentions the same variable twice, $a = R(y, y)$ for some $y \in \{x_1, x_2\}$. Note that $\mathbf{ans} = Mat(ans)$ will be of column vector type $(\tau(y), 1)$ since $\overline{x} = (y)$. Also note that, because of well-typedness, $\mathbf{R}$ must be of type $(\tau(y), \tau(y))$, i.e., $\mathbf{R}$ is a square matrix. Then construct e as follows. First consider the expression $e' := \mathbf{I}^{\tau(y)} \odot \mathbf{R}$. This selects from $\mathbf{R}$ all the entries on the diagonal, and sets to $\mathbb{0}$ all other entries. Then take $e := e' \cdot \mathbf{1}^{\tau(y)}$, which converts the diagonal entries into a column vector.

- Atom a is a unary atom of the form $a = R(y)$ with $y \in \{x_1, x_2\}$. We distinguish two subcases.
  - $\mathbf{R}$ has column vector type. In that case, take $e := \mathbf{R}$.
  - $\mathbf{R}$ has row vector type. Because **ans** always has column vector type by definition, we convert the input row vector into a column vector by taking $e := \mathbf{R}^T$.
- Atom a is a nullary atom, $a = R()$. Then $\mathbf{R}$ necessarily has type 1×1, as does **ans**. So, it suffices to take $e := \mathbf{R}$.

Second, when ψ is comparison of the form $x \leq \alpha$ then the simulating expression e is $\mathbf{1}^\alpha$. This expression has type $(\tau(x), 1)$, which conforms to the desired type of the definition since $\tau(x) = \alpha$.

Third, when $\psi = \exists y.\psi_1$ we may assume w.l.o.g. that $y \in \mathit{free}(\psi_1)$ since otherwise $\psi \equiv \psi_1$ and the result follows directly from the induction hypothesis. So, assume $y \in \mathit{free}(\psi_1) \subseteq \{x_1, x_2\}$. There are two cases.

- If $\mathit{free}(\psi_1) = \{x_1, x_2\}$ then $e_1 \colon (\tau(x_1), \tau(x_2))$ simulates ψ_1. If $y = x_2$ then we take $e = e_1 \cdot \mathbf{1}^{\tau(y)}$ to simulate ψ w.r.t. *Mat*. If $y = x_1$ then we take $e = e_1^T \cdot \mathbf{1}^{\tau(y)}$ to simulate ψ w.r.t. *Mat*. Note that in both cases the simulating expression has type $(\tau(x), 1)$ with x the unique variable in $\{x_1, x_2\} \setminus \{y\}$, as desired.
- If $\mathit{free}(\psi_1) = \{y\}$, then $e_1 \colon (\tau(y), 1)$ simulates ψ_1 and thus $e_1^T \cdot \mathbf{1}^{\tau(y)} \colon (1, 1)$ simulates ψ w.r.t. *Mat*.

The fourth and final case is when $\psi = \psi_1 \wedge \psi_2$. By inductive hypothesis there exists fc-MATLANG expressions e_1, e_2 that simulate ψ_1 and ψ_2 w.r.t. *Mat*, respectively. If $\mathit{free}(\psi_1) = \mathit{free}(\psi_2)$ then the simulating expression is simply $e_1 \odot e_2$. It will be of type $(\tau(x_1), \tau(x_2))$, $(\tau(x_1), 1)$, $(\tau(x_2), 1)$ or $(1, 1)$ depending if the free variables are $\{x_1, x_2\}$, $\{x_1\}$, $\{x_2\}$ or $\emptyset$, respectively. When $\mathit{free}(\psi_1) \neq \mathit{free}(\psi_2)$, we make the following case distinction.

(1) If $\mathit{free}(\psi_1) = \{x_1, x_2\}$ then $e_1 \colon (\tau(x_1), \tau(x_2))$.
   (a) If $\mathit{free}(\psi_2) = \{x_1\}$ then $e_2 \colon (\tau(x_1), 1)$ and we take $e := e_1 \odot \left(e_2 \cdot \left(\mathbf{1}^{\tau(x_2)}\right)^T\right)$.
   (b) If $\mathit{free}(\psi_2) = \{x_2\}$ then $e_2 \colon (\tau(x_2), 1)$ and we take $e := e_1 \odot \left(\mathbf{1}^{\tau(x_1)} \cdot e_2^T\right)$.
   (c) If $\mathit{free}(\psi_2) = \emptyset$ then $e_2 \colon (1, 1)$ and the simulating expression is $e_2 \times e_1$.

(2) If $\mathit{free}(\psi_1) = \{x_1\}$ then $e_1 \colon (\tau(x_1), 1)$.
   (a) If $\mathit{free}(\psi_2) = \{x_1, x_2\}$ then we reason analogous to case (1)(b).
   (b) If $\mathit{free}(\psi_2) = \{x_2\}$ then $e_2 \colon (\tau(x_2), 1)$ and define $e := \left(e_1 \cdot \left(\mathbf{1}^{\tau(x_2)}\right)^T\right) \odot \left(\mathbf{1}^{\tau(x_1)} \cdot e_2^T\right)$.
   (c) If $\mathit{free}(\psi_2) = \emptyset$ then $e_2 \colon (1, 1)$ and the simulating expression is $e_2 \times e_1$.

(3) If $\mathit{free}(\psi_1) = \{x_2\}$ we reason analogous to case (2).

(4) If $\mathit{free}(\psi_1) = \emptyset$ then $\mathit{free}(\psi_2) \neq \emptyset$ and we reason analogous to the cases above where $\mathit{free}(\psi_2) = \emptyset$ but $\mathit{free}(\psi_1) \neq \emptyset$. □

Finally, we extend Proposition 6.3 from formulas to queries.

Corollary 6.4. *Let Q be an $\mathsf{FO}_2^\wedge$ query over σ and let Mat be a relational-to-matrix schema encoding on $\sigma(Q)$ such that $Mat \vdash Q \colon \tau$. Then there exists* fc-MATLANG *query* $\mathbf{Q}$ *that simulates Q w.r.t. Mat.*

Proof. Let $Q \colon H \leftarrow \psi$ be a $\mathsf{FO}_2^\wedge$ query over σ and let *Mat* be a relational-to-matrix schema encoding on $\sigma(Q)$ such that $Mat \vdash \psi \colon \tau$. By Proposition 6.3 there exists $e \in$ fc-MATLANG that simulates ψ w.r.t. *Mat*. Let $Mat(H) = \mathbf{H}$. We derive a fc-MATLANG query $\mathbf{Q} \colon \mathbf{H} := \overline{e}$ that simulates Q as follows.

- When $\overline{x} = (x, y)$, if $x \prec y$ then define $\overline{e} := e$, otherwise define $\overline{e} := e^T$.
- When $\overline{x} = (x, x)$ then Q outputs the encoding of a square matrix, where the value of diagonal entry (x, x) is computed by $\psi(x)$. To simulate this query, we observe that $e\colon (\tau(x), 1)$ is a column vector that simulates ψ. We first compute the square matrix of type $(\tau(x), \tau(x))$ with the vector e in all its columns as $e \cdot \left(\mathbf{1}^{\tau(x)}\right)^T$. Then set all non-diagonal entries of the former expression to zero, by taking $\overline{e} := \mathbf{I}^{\tau(x)} \odot \left(e \cdot \left(\mathbf{1}^{\tau(x)}\right)^T\right)$.
- When $\overline{x} = (x)$ we have $e\colon (\tau(x), 1)$. If $\mathbf{H}\colon (\alpha, 1)$ define $\overline{e} := e$ otherwise if $\mathbf{H}\colon (1, \alpha)$ define $\overline{e} := e^T$.
- When $\overline{x} = ()$ we have $e\colon (1, 1)$. Then define $\overline{e} := e$. □

6.2. **From fc-MATLANG to $\mathsf{FO}^{\wedge}_2$.** To establish the converse direction of Theorem 6.1 we first define when an FO^+ *formula* simulates a sum-MATLANG sentence instead of a query. Let $e\colon (\alpha, \beta)$ be a sum-MATLANG sentence over $\mathcal{S}$ and let *Rel* be a matrix-to-relational schema encoding on $\mathcal{S}$. We say that FO^+ formula ψ *simulates* e *w.r.t. Rel* if ψ has exactly two free variables $x \prec y$ and for every matrix instance $\mathcal{I}$ and all i, j we

$$[\![\psi]\!]_{Rel(\mathcal{I})}(x \mapsto i, y \mapsto j) = \begin{cases} [\![e]\!](\mathcal{I})_{i,j} & \text{if } 1 \leq i \leq \mathcal{I}(\alpha), 1 \leq j \leq \mathcal{I}(\beta) \\ \mathbb{0} & \text{otherwise} \end{cases}.$$

Let $\mathsf{FO}^{\wedge}_{2,=}$ denote the two-variable fragment of $\mathsf{FO}^{\wedge}$ where equality atoms are allowed (subject to being safe), as long as only two variables are used in the entire formula. For proving the direction from fc-MATLANG to $\mathsf{FO}^{\wedge}_2$, we first do the proof by using equality, and later show how to remove it.

Proposition 6.5. *Let e be a* fc-MATLANG *expression over $\mathcal{S}$ and let Rel be a matrix-to-relational schema encoding on $\mathcal{S}$. There exists $\psi_e \in \mathsf{FO}^{\wedge}_{2,=}$ that simulates e w.r.t. Rel.*

Proof. Let $e \in$ fc-MATLANG and fix two distinct variables $x \prec y$. We build the formula $\psi_e \in \mathsf{FO}^{\wedge}_{2,=}$ that simulates e w.r.t. *Rel* by induction on e. In particular, we build ψ_e such that $\mathit{free}(\psi_e) = \{x, y\}$. For a matrix symbol $\mathbf{A}$, let us denote $\mathit{Rel}(A)$ simply as A.

(1) If $e = \mathbf{A}$ then take:
   - $\psi_e \coloneqq A(x, y)$ when $\mathtt{type}(\mathbf{A}) = (\alpha, \beta)$; $\mathtt{type}(\mathbf{A}) = (1, 1)$; A is binary and $\mathtt{type}(\mathbf{A}) = (\alpha, 1)$ or $\mathtt{type}(\mathbf{A}) = (1, \beta)$.
   - $\psi_e \coloneqq A(x) \wedge y \leq 1$ if A is unary and $\mathtt{type}(\mathbf{A}) = (\alpha, 1)$.
   - $\psi_e \coloneqq x \leq 1 \wedge A(y)$ if A is unary and $\mathtt{type}(\mathbf{A}) = (1, \beta)$.

(2) If $e = \mathbf{1}^{\alpha}\colon (\alpha, 1)$ then take $\psi_e \coloneqq x \leq \alpha \wedge y \leq 1$.

(3) If $e = \mathbf{I}^{\alpha}\colon (\alpha, \alpha)$ then take $\psi_e \coloneqq x \leq \alpha \wedge y \leq \alpha \wedge x = y$. This will be the only case where we include an equality atom.

(4) If $e = (e')^T\colon (\beta, \alpha)$ then take $\psi_e \coloneqq \psi_{e'}[x \leftrightarrow y]$ where $\psi_{e'}[x \leftrightarrow y]$ denotes the formula obtained from $\psi_{e'}$ by swapping x and y. (I.e., simultaneously replacing all occurrences of x—free and bound—with y and all occurrences of y with x.)

(5) If $e = e_1 \times e_2$ with $e_1\colon (1, 1)$ and $e_2\colon (\alpha, \beta)$ we reason as follows. First observe that because ψ_{e_1} simulates e_1 and because e_1 always outputs scalar matrices, $[\![\psi_{e_1}]\!]_{Rel(\mathcal{I})}(x \mapsto i, y \mapsto j) = \mathbb{0}$ whenever $i \neq 1$ or $j \neq 1$. We conclude that therefore, $\exists x, y.\psi_{e_1}$ will always return $[\![e_1]\!](\mathcal{I})(1, 1)$ when evaluated on $\mathit{Rel}(\mathcal{I})$. Hence, we take $\psi_e \coloneqq (\exists x, y.\psi_{e_1}) \wedge \psi_{e_2}$.

(6) If $e = e_1 \odot e_2$ then take $\psi_e \coloneqq \psi_{e_1} \wedge \psi_{e_2}$.

(7) If $e = e_1 \cdot v_2$ with $e_1 \colon (\alpha, \beta)$ and with $v_2 \colon (\beta, 1)$ evaluating to a column vector, we reason as follows. First observe that because ψ_{v_2} simulates v_2 and because v_2 always outputs column vectors, $[\![\psi_{v_2}]\!]_{Rel(\mathcal{I})}(x \mapsto i, y \mapsto j) = \mathbb{0}$ whenever $j \neq 1$. Let $\varphi_{v_2} = \psi_{v_2}[x \leftrightarrow y]$ be the $\mathsf{FO}^{\wedge}_{2,=}$ formula obtained from ψ_{v_2} by simultaneously replacing all occurrences (free and bound) of x by y, and all occurrences of y by x. Then $[\![\varphi_{v_2}]\!]_{Rel(\mathcal{I})}(y \mapsto i, x \mapsto j) = \mathbb{0}$ whenever $j \neq 1$. We conclude that therefore

$$[\![v_2]\!](\mathcal{I})_{i,1} = [\![\exists x.\varphi_{v_2}]\!]_{Rel(\mathcal{I})}(y \mapsto i) \text{ for all } \mathcal{I} \text{ and } i. \tag{6.1}$$

Next observe that e also always outputs a column vector, and that

$$[\![e]\!](\mathcal{I})_{i,1} = [\![e_1 \cdot v_2]\!](\mathcal{I})_{i,1} = \bigoplus_k [\![e_1]\!](\mathcal{I})_{i,k} \odot [\![v_2]\!](\mathcal{I}, \mu)_{k,1}$$

Take $\psi' = \exists y. \big(\psi_{e_1}(x, y) \wedge \varphi'_{v_2}(y)\big)$ where $\varphi'_{v_2} = \exists x.\varphi_{v_2}$. Then ψ' is in $\mathsf{FO}^{\wedge}_{2,=}$. Because ψ_{e_1} simulates e_1 and because of (6.1) it follows that $[\![e]\!](\mathcal{I})_{i,1} = [\![\psi']\!]_{Rel(\mathcal{I})}(x \mapsto i)$ for all $\mathcal{I}$ and i. Hence, we take $\psi_e := \psi' \wedge y \leq 1$ to simulate e.

(8) The case $e = v_1 \cdot e_2$ is similar to the previous case. □

To establish our desired result at the level of queries we require the following two lemmas that will help to remove the equalities from $\mathsf{FO}^{\wedge}_{2,=}$. The proofs can be found in Appendix C.

Lemma 6.6. *Let φ be a $\mathsf{FO}^{\wedge}_2$ formula (hence, without equality) using only the distinct variables x, y such that free$(\varphi) \cap \{x, y\} \neq \emptyset$. Let ψ be the $\mathsf{FO}^{\wedge}_{2,=}$ formula $\exists x.(\varphi \wedge x = y)$. Note that free$(\psi) = \{y\}$. There exists an $\mathsf{FO}^{\wedge}_2$ formula ψ' equivalent to ψ.*

Lemma 6.7. *For every $\mathsf{FO}^{\wedge}_{2,=}$ formula ψ there exists an equivalent $\mathsf{FO}^{\wedge}_{2,=}$ formula ψ' where equality atoms only occur at the top level: for every subformula $\exists z.\phi'$ of ψ' it holds that $\phi' \in \mathsf{FO}^{\wedge}_2$ does not contain equality atoms.*

Finally, we can prove the direction from fc-MATLANG to $\mathsf{FO}^{\wedge}_2$.

Corollary 6.8. *Let $\mathbf{Q}$ be an* fc-MATLANG *query over $\mathcal{S}$ and let Rel be a matrix-to-relational schema encoding on $\mathcal{S}$. There exists an $\mathsf{FO}^{\wedge}_2$ query Q that simulates $\mathbf{Q}$ w.r.t. Rel.*

Proof. Let $\mathbf{H} := e$ be an fc-MATLANG query and let $H = Rel(\mathbf{H})$. By Proposition 6.5 there exists $\psi \in \mathsf{FO}^{\wedge}_{2,=}$ that simulates e w.r.t. *Rel*. By construction, ψ has two distinct free variables. Assume that x, y are these two distinct variables, and $x \prec y$. There are three cases to consider.

(1) H is a binary relation. By Lemma 6.7 there exists $\psi' \in \mathsf{FO}^{\wedge}_{2,=}$ that is equivalent to ψ and where equality atoms only occur at the top level. Because ψ has $\{x, y\}$ as free variables, so does ψ'. If no equality atom occurs in ψ' then we take Q to be the query $H(x, y) \leftarrow \psi'$. If some equality atom does occur in ψ' then we may observe w.l.o.g. that the equality is $x = y$ since the equalities $x = x$ and $y = y$ are trivial and can always be removed. As such, $\psi' \equiv \psi'' \wedge x = y$ with $\psi'' \in \mathsf{FO}^{\wedge}_2$ and at least one of x, y appearing free in ψ''. Assume w.l.o.g. that x occurs free in ψ''. Then the query $H(x, x) \leftarrow \exists y.\psi'$ simulates $\mathbf{Q}$ w.r.t. *Rel*. Although this is formally a query in $\mathsf{FO}^{\wedge}_{2,=}$, we can we can next apply Lemma 6.6 to ψ'' to obtain an equivalent formula in $\mathsf{FO}^{\wedge}_2$ for the body $\exists y.\psi' \equiv \exists y.(\psi'' \wedge x = y)$, yielding the desired result.

(2) H is a unary relation. Assume that $\mathbf{H}$ is of column vector type $(\alpha, 1)$; the reasoning when $\mathbf{H}$ has row vector type is similar. Take $\varphi = \exists y.\psi$. Because ψ simulates e and because e always outputs column vectors, $[\![\psi]\!]_{Rel(\mathcal{I})}(x \mapsto i, y \mapsto j) = \mathbb{0}$ whenever $j \neq 1$, for every $\mathcal{I}$. We conclude that therefore $[\![e]\!](\mathcal{I})(i, 1) = [\![\exists y.\psi]\!]_{Rel(\mathcal{I})}(x \mapsto i)$ for all i and $\mathcal{I}$. As such,

the query $Q'\colon H(x) \leftarrow \varphi$ simulates $\mathbf{Q}$. While Q' is a query in $\mathsf{FO}^{\wedge}_{2,=}$, we can next apply Lemma 6.7 to φ to obtain an equivalent formula $\varphi' \equiv \varphi$ in which equality only occurs at the top level. Note that, since φ has only $\{x\}$ as free variables, so does φ'. Hence, any equality that occurs at the top level in φ' must be of the form $x = x$, which is trivial and can therefore be removed. After removing such trivial equalities, take $Q\colon H(x) \leftarrow \varphi'$ as the simulating query.

(3) H is a nullary relation. The reason is entirely similar to the previous case. □

7. Q-hierarchical conjunctive queries

We next turn our attention to specializing the correspondence between conj-MATLANG and CQs given by Corollary 4.7 to *q-hierarchical* CQs [BDG07, Bra13]. Q-hierarchical CQs are relevant since, over the Boolean semiring, they capture 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 [BKS17]. Formally, q-hierarchical CQs are defined as follows. 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 from the body of Q 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 hold:

(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.

In this section, we first prove a correspondence between qh-CQs and a fragment of $\mathsf{FO}^{\wedge}_2$. Using this, we characterize the fragment of MATLANG that corresponds to qh-CQs in Section 8. Concretely, we prove the following result in this section. We say that a $\mathsf{FO}^{\wedge}_2$ formula φ is *simple* if every subformula of the form $\varphi_1 \wedge \varphi_2$ in φ satisfies $free(\varphi_1) = free(\varphi_2)$. We denote by $\mathsf{simple\text{-}FO}^{\wedge}_2$ the class of all simple $\mathsf{FO}^{\wedge}_2$ formulas. Let $\varphi_1, \ldots, \varphi_k \in \mathsf{simple\text{-}FO}^{\wedge}_2$ with $k \geq 1$ such that $free(\varphi_i) \subseteq \{x, y\}$ for two distinct variables $\{x, y\}$. We say that $\varphi_1 \wedge \ldots \wedge \varphi_k$ is a *hierarchical conjunction* if $\{\{x, y\}, \{x\}, \{y\}\} \not\subseteq \{free(\varphi_i) \mid 1 \leq i \leq n\}$. For example, $R(x, y) \wedge S(y) \wedge U()$ is a hierarchical conjunction, but $R(x, y) \wedge S(y) \wedge T(x)$ is not. We denote by $\mathsf{h\text{-}FO}^{\wedge}_2$ all hierarchical conjunctions of $\mathsf{simple\text{-}FO}^{\wedge}_2$ formulas. Observe that, by definition, $\mathsf{simple\text{-}FO}^{\wedge}_2 \subseteq \mathsf{h\text{-}FO}^{\wedge}_2$. As usual, an $\mathsf{h\text{-}FO}^{\wedge}_2$ query is a query with body ψ such that $\psi \in \mathsf{h\text{-}FO}^{\wedge}_2$.

Theorem 7.1. *Q-hierarchical binary CQs and* $\mathsf{h\text{-}FO}^{\wedge}_2$ *queries are equally expressive.*

To prove Theorem 7.1 we first define *guarded query plans* in Section 7.1. Similar to how query plans allowed us to prove equivalence between fc-CQs and $\mathsf{FO}^{\wedge}_2$ in Section 5, we use guarded query plans to prove equivalence between qh-CQs and $\mathsf{h\text{-}FO}^{\wedge}_2$. We show that $\mathsf{h\text{-}FO}^{\wedge}_2$ is included in qh-CQs in Section 7.2, and the converse in Section 7.3.

7.1. **Guarded query plans.** A generalized join tree (GJT) T (see Section 5.2) is *guarded* if for every node n and every child c of n it holds that $var(n) \subseteq var(c)$, i.e., in a guarded GJT every child is a guard of its parent. A query plan (QP) is guarded if its GJT is guarded.[2] The following proposition is a particular case of the results shown in [IUV17, IUV$^+$20].

Proposition 7.2. *A CQ Q is q-hierarchical if and only if it has a guarded query plan.*

For later use we observe the following property. Its proof is completely analogous to the proof of Lemma 5.5.

Lemma 7.3. *For every guarded QP there exists an equivalent guarded QP such that for any node n with two children c_1, c_2 it holds that $var(n) = var(c_1) = var(c_2)$.*

7.2. **From q-hierarchical CQs to h-$\mathsf{FO}_2^\wedge$ queries.**

Lemma 7.4. *For every guarded QP (T, r) such that $\varphi[T, r]$ is binary, there exists a simple-$\mathsf{FO}_2^\wedge$ formula φ equivalent to $\varphi[T, r]$.*

Proof. By Lemma 7.3 we may assume w.l.o.g. that T is such that for every node n with two children c_1, c_2 it holds $var(c_1) = var(n)$ and $var(c_2) = var(n)$. In the proof of Proposition 5.11 we showed by induction on T that we can construct an $\mathsf{FO}_2^\wedge$ formula φ equivalent to $\varphi[T, r]$. It is readily verified that when T is guarded this construction yields a simple formula. □

Proposition 7.5. *For every guarded QP (T, N) such that $\varphi[T, N]$ is binary, there exists an h-$\mathsf{FO}_2^\wedge$ formula ψ equivalent to $\varphi[T, N]$.*

Proof. By Lemma 7.3 we may assume w.l.o.g. that T is such that for every node n with two children c_1, c_2 we have $var(c_1) = var(c_2) = var(n)$. Let $F = \{r_1, \ldots, r_l\}$ be the frontier of N. Let $T_1, \ldots, T_l$ be the subtrees of T rooted at $r_1, \ldots, r_l$, respectively. By Lemma 7.4, there are simple-$\mathsf{FO}_2^\wedge$ formulas $\psi_1, \ldots, \psi_l$ equivalent to $\varphi[T_1, r_1], \ldots, \varphi[T_l, r_l]$, respectively. By Lemma 5.7, $\varphi[T, N] \equiv \varphi[T_1, r_1] \wedge \cdots \wedge \varphi[T_l, r_l] \equiv \psi_1 \wedge \cdots \wedge \psi_l$. Because $\varphi[T, N]$ is binary, there exists a set $\{x, y\}$ of two distinct variables such that $var(N) \subseteq \{x, y\}$. It follows that $free(\psi_i) = var(r_i) \subseteq var(N) \subseteq \{x, y\}$ for every i. It remains to show that $\psi_1 \wedge \cdots \wedge \psi_l$ is a hierarchical conjunction. To that end, assume, for the purpose of obtaining a contradiction, that $\{\{x, y\}, \{x\}, \{y\}\} \subseteq \{free(\psi_i) \mid 1 \leq i \leq n\}$. Because $free(\psi_i) = var(r_i)$ for every i there must exist nodes $r_{x,y}$, r_x and r_y in F that are labeled by $\{x, y\}$, $\{x\}$ and $\{y\}$ respectively. Now note that, because T is guarded and satisfies Lemma 7.3, all ancestors of r_x (resp. r_y) can only contain x (resp. y) in their label, but not y (resp x). At the same time, because of the connectedness property, there is an ancestor a of both $r_{x,y}$ and r_x that must contain x, implying that $var(a) = \{x\}$. Similarly, there is an ancestor b of both $r_{x,y}$ and r_y that must contain y, implying that $var(b) = \{y\}$. Now note that a and b are both ancestors of $r_{x,y}$, so either a is an ancestor of b, or the converse holds. Assume a is an ancestor of b. But then a is also an ancestor of r_y and a has x in its label, which yields the desired contradiction. The case where b is an ancestor of a is similar. □

[2] Guarded GJTs and QPs are called *simple* in [IUV$^+$20].

7.3. From h-FO$_2^\wedge$ queries to q-hierarchical CQs.

Lemma 7.6. *For every* simple-FO$_2^\wedge$ *formula* φ *there exists a guarded QP* (T, r) *such that* $\varphi[T, r]$ *is binary and equivalent to* φ*.*

Proof. The proof is analogous to the proof of Proposition 5.13. Fix two distinct variables x and y and formula $\varphi \in$ simple-FO$_2^\wedge$. Because FO$_2^\wedge$ formulas use at most two variables we may assume w.l.o.g. that the only variables occurring in φ are x and y. We prove by induction on φ that there exists a guarded query plan (T, r) such that $\varphi[T, r] \equiv \varphi$.

When φ is an atom (relational or inequality) then the result trivially follow by creating a GJT T with single node r, labeled by the atom, and fixing $N = \{r\}$.

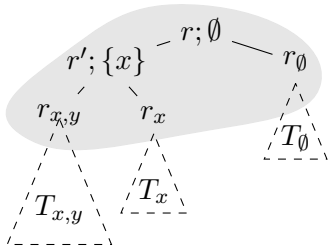
When $\varphi = \exists z.\varphi'$ with $z \in \{x, y\}$ we may assume w.l.o.g. that $z \in \mathit{free}(\varphi')$: if not then $\varphi \equiv \varphi'$ and the result follows directly from the induction hypothesis on φ'. So, assume $z \in \mathit{free}(\varphi)$. By induction hypothesis we have a guarded and binary QP (T', r') equivalent to φ' with r' the root of T'. Let T be obtained by adjoining a new root r to T'; label r by $\mathit{var}(r') \setminus \{z\}$; and make r' the only child of r. Take $N = \{r\}$. Then (T, N) remains binary and guarded. Equivalence of $\varphi[T, N]$ to ψ follows by induction hypothesis and Lemma 5.8.

Otherwise φ is a conjunction, $\varphi = \varphi_1 \wedge \varphi_2$. By induction hypothesis, we have binary and guarded QPs (T_1, r_1) and (T_2, r_2) equivalent to ψ_1 and ψ_2, respectively with r_1 and r_2 the roots of T_1, T_2. Using the same reasoning as in the proof of Proposition 5.13 we may assume w.l.o.g. that T_1 and T_2 have no nodes in common, and that moreover $\mathit{var}(T_1) \cap \mathit{var}(T_2) \subseteq \mathit{var}(N_1) \cap \mathit{var}(N_2)$, i.e., that all variables that are mentioned in both T_1 and T_2 are also in $\mathit{var}(N_1) \cap \mathit{var}(N_2)$. Because (T_i, r_i) is equivalent to φ_i we have $\mathit{var}(r_i) = \mathit{free}(\varphi_i)$, for $i = 1, 2$. Then, because φ is simple, we necessarily have $\mathit{var}(r_1) = \mathit{free}(\varphi_1) = \mathit{free}(\varphi_2) = \mathit{var}(r_2)$. As such, $\mathit{var}(r_1) = \mathit{var}(r_2)$. We construct the claimed QP (T, r) by taking the disjoint union of T_1 and T_2 and adjoining a new root r with children r_1, r_2 such that $\mathit{var}(r) = \mathit{var}(r_1) = \mathit{var}(r_2)$. It is readily verified that T satisfies the connectedness property and is hence a GJT. Moreover, T is guarded. Correctness follows by Corollary 5.9. □

Proposition 7.7. *For every* h-FO$_2^\wedge$ *formula* ψ *there exists a guarded QP* (T, N) *such that* $\varphi[T, N]$ *is binary and equivalent to* ψ*.*

Proof. Let x, y be distinct FO variables, let $\psi = \varphi_1 \wedge \cdots \wedge \varphi_k \in$ h-FO$_2^\wedge$ with $\varphi_i \in$ simple-FO$_2^\wedge$ and $\mathit{free}(\varphi_i) \subseteq \{x, y\}$ for $i = 1, \ldots, k$. Consider the set $F = \{\mathit{free}(\varphi_i) \mid 1 \leq i \leq k\}$. Note that $F \neq \emptyset$ since it includes the root. Furthermore, because ψ is a hierarchical conjunction, $\{\{x, y\}, \{x\}, \{y\}\} \not\subseteq F$. Since $F \neq \emptyset$, there are hence $\binom{4}{1} + \binom{4}{2} + \left(\binom{4}{3} - 1\right) = 19$ possibilities for F. We construct (T, N) by case analysis on F. We only illustrate one case; the other cases are similar.

Assume $F = \{\{x, y\}, \{x\}, \emptyset\}$. Define, for each $S \in F$ the formula φ_S to be the conjunction of all φ_i with $\mathit{free}(\varphi_i) = S$. Then $\psi \equiv \varphi_{x,y} \wedge \varphi_x \wedge \varphi_\emptyset$. Observe that each φ_S is simple. Hence by Proposition 7.6 there exist guarded query plans $(T_{x,y}, r_{x,y}), (T_x, r_x)$ and $(T_\emptyset, r_\emptyset)$ equivalent to $\varphi_{x,y}, \varphi_x, \varphi_\emptyset$, respectively. In particular, $\mathit{var}(r_{x,y}) = \{x, y\}$, $\mathit{var}(r_x) = \{x\}$ and $\mathit{var}(r_\emptyset) = \emptyset$. We may assume w.l.o.g. that the GJTs $T_{x,y}, T_x$ and $T_\emptyset$ do not have nodes in common. We construct the claimed guarded QP (T, N) by constructing T as follows and setting $N = \{r, r', r_{x,y}, r_x, r_\emptyset\}$.



Here, r and r' are new nodes with $var(r) = \emptyset$; $var(r') = \{x\}$. It is readily verified that T is guarded. Correctness follows from Lemma 5.7 since $\{r_{x,y}, r_x, r_\emptyset\}$ is the frontier of N. □

8. The q-hierarchical fragment of matlang queries

We next define qh-MATLANG, a fragment of sum-MATLANG that we will show to be equally expressive as h-$\mathsf{FO}_2^\wedge$ and qh-CQs. Like h-$\mathsf{FO}_2^\wedge$, qh-MATLANG is a two-layered language where expressions in the top layer can only be built from the lower layer. This lower layer, called simple-MATLANG, is a fragment of fc-MATLANG defined by

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

Note in particular that while fc-MATLANG allows arbitrary matrix-vector multiplication, this is restricted to multiplication with the ones vector in simple-MATLANG. Intuitively, simple-MATLANG 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 also q-hierarchical. For this reason, we enhance simple-MATLANG with the higher layer. Specifically, let us call expressions of the form $e \cdot (\mathbf{1}^\alpha)^T$ and $\mathbf{1}^\alpha \cdot e$ *expansions* of simple-MATLANG expression e. Note that these are well-typed only if e has column resp. row vector type. These expressions construct a matrix from a column (resp. row) vector by duplicating the column (resp. row) α times. We then define qh-MATLANG to consist of all expressions g of the form

$$\begin{aligned} f &::= \quad e \mid e \cdot (\mathbf{1}^\alpha)^T \mid \mathbf{1}^\alpha \cdot e \\ g &::= \quad f \mid f \odot f \end{aligned}$$

where e ranges over simple-MATLANG expressions, and f over simple-MATLANG expressions and expansions thereof.

In this section we prove:

Theorem 8.1. qh-MATLANG *queries and* h-$\mathsf{FO}_2^\wedge$ *queries are equally expressive.*

Combined with Theorem 7.1 we hence obtain.

Corollary 8.2. qh-MATLANG *and q-hierarchical binary CQs are equally expressive.*

The rest of this section is devoted to proving Theorem 8.1. We show that qh-CQs are at least as expressive as qh-MATLANG in Section 8.1 and the converse in Section 8.2.

8.1. **From h-FO$_2^\wedge$ formulas to qh-MATLANG expressions.** We will first simulate h-FO$_2^\wedge$ *formulas* by means of a qh-MATLANG expression according to the definition in Section 6.1.

Proposition 8.3. *Let $\varphi \in$ simple-FO$_2^\wedge$ be a formula over σ. Let Mat be a relational-to-matrix schema encoding on σ such that $Mat \vdash \varphi\colon \tau$. Then there exists a* simple-MATLANG *expression e that simulates φ w.r.t. Mat.*

Proof. We assume $var(\varphi) \subseteq \{x, y\}$ with $x \prec y$. The proof is analogous to the proof of Proposition 6.3, with the difference that when $\varphi = \varphi_1 \wedge \varphi_2$ the simulating expression is always $e_1 \odot e_2$ because $free(\varphi_1) = free(\varphi_2)$ since $\varphi \in$ simple-FO$_2^\wedge$. This has the consequence that the simulating expression is always a simple-MATLANG expression. □

Proposition 8.4. *Let $\psi \in$ h-FO$_2^\wedge$ be a formula over σ. Let Mat be a relational-to-matrix schema encoding on σ such that $Mat \vdash \psi\colon \tau$. Then there exists a* qh-MATLANG *expression e that simulates ψ w.r.t. Mat.*

Proof. We assume $var(\psi) \subseteq \{x, y\}$ with $x \prec y$. By definition, $\psi = \varphi_1 \wedge \cdots \wedge \varphi_k$ with $\varphi_i \in$ simple-FO$_2^\wedge$ for every i. Consider the set $F = \{free(\varphi_i) \mid 1 \leq i \leq k\}$. Note that $F \neq \emptyset$. Furthermore, because ψ is a hierarchical conjunction, $\{\{x, y\}, \{x\}, \{y\}\} \not\subseteq F$. Since $F \neq \emptyset$, there are hence $\binom{4}{1} + \binom{4}{2} + \left(\binom{4}{3} - 1\right) = 19$ possibilities for F. We construct e by case analysis on F. We only illustrate one case; the other cases are similar.

Assume $F = \{\{x, y\}, \{x\}, \emptyset\}$. Define, for each $S \in F$ the formula φ_S to be the conjunction of all φ_i with $free(\varphi_i) = S$. Then $\psi \equiv \varphi_{x,y} \wedge \varphi_x \wedge \varphi_\emptyset$. Observe that each φ_S is simple. Hence by Proposition 8.4 there exist simple-MATLANG expressions $e_{x,y}, e_x$, and $e_\emptyset$ that simulate $\varphi_{x,y}, \varphi_x$, and $\varphi_\emptyset$, respectively. Then the qh-MATLANG expression simulating φ is

$$e \coloneqq \left(e_{\varphi_\emptyset} \times e_{\varphi_{\{x,y\}}}\right) \odot \left(e_{\varphi_{\{x\}}} \cdot \left(\mathbf{1}^{\tau(y)}\right)^T\right). \qquad \square$$

To also establish the correspondence at the level of queries we first observe the following straightforward fact. For a sum-MATLANG expression $e\colon (\alpha, 1)$ define $\mathsf{diag}(e)$ to return a matrix of type (α, α) where the entries of e are stored in the diagonal and where all non-diagonal entries are $\mathbb{0}$.

Lemma 8.5. qh-MATLANG *is closed under transpose and diagonalisation: for any expression $g \in$ qh-MATLANG also g^T is expressible in* qh-MATLANG *and, moreover, if $g\colon (\alpha, 1)$ has column vector type, then $\mathsf{diag}(g)$ is expressible in* qh-MATLANG.

Proof. Closure under transpose follows from the fact that simple-MATLANG is closed under transpose by definition, and from the following equivalences:

$$(f_1 \odot f_2)^T = f_1^T \odot f_2^T \qquad \left(e \cdot (\mathbf{1}^\alpha)^T\right)^T = \mathbf{1}^\alpha \cdot e^T \qquad (\mathbf{1}^\alpha \cdot e)^T = e^T \cdot (\mathbf{1}^\alpha)^T$$

Closure under diagonalisation is straightforward. Assume that $g\colon (\alpha, 1)$. Note that expansions never have column vector type. As such, g is a simple-MATLANG expression, or a Hadamard product of simple-MATLANG expressions. Note that in the latter case, this is itself also a simple-MATLANG expression since simple-MATLANG is closed under Hadamard products. Then $\mathsf{diag}(g)$ can be define in qh-MATLANG by first computing the column expansion of g and subsequently setting all non-diagonal entries of this expansion to zero by taking

$$\mathsf{diag}(g) = \mathbf{I}^{\tau(x)} \odot \left(g \cdot \left(\mathbf{1}^{\tau(x)}\right)^T\right). \qquad \square$$

Corollary 8.6. *Let Q be an* h-$\mathsf{FO}_2^\wedge$ *query over σ and let Mat be a relational-to-matrix schema encoding on $\sigma(Q)$ such that $Mat \vdash Q : \tau$. Then there exists* qh-MATLANG *query* $\mathbf{Q}$ *that simulates Q w.r.t. Mat.*

Proof. Assume $Q : H(\overline{x}) \leftarrow \psi$. By Proposition 8.4 there exists a qh-MATLANG expression e that simulates ψ w.r.t. *Mat*. Let $Mat(H) = \mathbf{H}$. We build the qh-MATLANG query $\mathbf{Q} := \mathbf{H} = \overline{e}$ as follows, using Lemma 8.5:

- When $\overline{x} = (x, y)$, if $x \prec y$ then define $\overline{e} := e$, otherwise define $\overline{e} := e^T$.
- When $\overline{x} = (x, x)$ we note that $free(\psi) = \{x\}$ and therefore $e : (\tau(x), 1)$ outputs the encoding of a square matrix, where the value of diagonal entry (x, x) is computed by $\psi(x)$. All non-diagonal entries of this matrix are $\mathbb{0}$. Hence we take $\overline{e} := \mathsf{diag}(e)$.
- When $\overline{x} = (x)$ we have $e : (\tau(x), 1)$. If $\mathbf{H} : (\tau(x), 1)$ define $\overline{e} := e$ otherwise if $\mathbf{H} : (1, \tau(x))$ define $\overline{e} := e^T$.
- When $\overline{x} = ()$ we have $e : (1, 1)$ and define $\overline{e} := e$. □

8.2. **From qh-MATLANG expressions to h-$\mathsf{FO}_2^\wedge$ formulas.** To establish the converse direction of Theorem 8.1 we first show that how to simulate qh-MATLANG expressions by means of h-$\mathsf{FO}_2^\wedge$ formulas that may also use equality, according to the definition in Section 6.2. Define h-$\mathsf{FO}_{2,=}^\wedge$ to consist of all h-$\mathsf{FO}_2^\wedge$ formulas as well as all formulas of the form $\varphi \wedge x = y$ or $x = y \wedge \varphi$ with $\varphi \in$ h-$\mathsf{FO}_2^\wedge$ and $free(\varphi) \cap \{x, y\} \neq \emptyset$. Note that in the latter case we assume that x, y are the only two distinct variables that can be used in φ, so that the result again uses only two variables.

Define the *signature* $\mathrm{sig}(\varphi)$ of formula $\varphi \in$ h-$\mathsf{FO}_{2,=}^\wedge$ to be the set $\{free(\varphi_i) \mid 1 \leq i \leq n\}$ where $\varphi_1, \ldots, \varphi_n$ are the non-equality conjuncts of φ. For example, the signature of $\varphi = x \leq c \wedge y \leq d \wedge x = y$ is $\{\{x\}, \{y\}\}$.

The following lemma shows that projections on h-$\mathsf{FO}_{2,=}^\wedge$ formula are expressible as h-$\mathsf{FO}_2^\wedge$ formulas (hence, not using equality), provided that the original formulas signature is of a certain shape.

Lemma 8.7. *Let φ be a* h-$\mathsf{FO}_{2,=}^\wedge$ *formula such that* $\mathrm{sig}(\varphi)$ *is a subset of either $\{\emptyset, \{x\}, \{y\}\}$ or $\{\emptyset, \{x, y\}\}$. There exists a* h-$\mathsf{FO}_2^\wedge$ *formula equivalent to $\exists x.\varphi$ (resp. $\exists y.\varphi$) whose signature is a subset of $\{\emptyset, \{x\}, \{y\}\}$.*

Proof. We first make the following claim: if $\psi \in$ simple-$\mathsf{FO}_2^\wedge$ and $free(\psi) \cap \{x, y\} \neq \emptyset$, then there exists a simple-$\mathsf{FO}_2^\wedge$ formula equivalent to $\exists x.(\varphi \wedge x = y)$ (resp. $\exists y.(\varphi \wedge x = y)$). The proof of this claim is identical to the proof of Lemma 6.6, observing that the construction defined there, when applied to simple formulas, yields a simple formula. Because the resulting formula has at most 1 free variable, its signature (viewed as a conjunction with only one conjunct) is trivially a subset of $\{\emptyset, \{x\}, \{y\}\}$.

Next, assume that $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_k \wedge x = y$ where the φ_i are simple-$\mathsf{FO}_2^\wedge$ formulas. (The reasoning when φ does not contain the equality atom $x = y$ is similar.) We make a case analysis on $\mathrm{sig}(\varphi)$.

- $\mathrm{sig}(\varphi)$ is a subset of $\{\emptyset, \{x\}, \{y\}\}$. Since φ has at least one conjunct, $\mathrm{sig}(\varphi)$ is non-empty. Moreover, the case where $\mathrm{sig}(\varphi) = \{\emptyset\}$ cannot occur, since otherwise φ would not be safe. There are hence $2^3 - 2 = 6$ possibilities to consider for the signature. We only illustrate the reasoning when $\mathrm{sig}(\varphi) = \{\emptyset, \{x\}, \{y\}\}$, the other cases are similar. Assume that $\mathrm{sig}(\varphi) = \{\emptyset, \{x\}, \{y\}\}$. For each $S \in \{\emptyset, \{x\}, \{y\}\}$ let ψ_S be the conjunction of all the

formulas φ_i with $\mathit{free}(\varphi_i) = S$. Note that each ψ_S is in $\mathsf{simple\text{-}FO}^{\wedge}_2$, as it is a conjunction of simple formulas with the same set of free variables, and that $\varphi \equiv \psi_\emptyset \wedge \psi_x \wedge \psi_y \wedge x = y$. Then $\exists x.\varphi \equiv \psi_\emptyset \wedge \exists x.(\psi_x \wedge x = y) \wedge \psi_y$. By our earlier claim, $\exists x.(\psi_x \wedge x = y)$ is expressible as a $\mathsf{simple\text{-}FO}^{\wedge}_2$ formula ψ'_x. Because $\exists x.(\psi_x \wedge x = y)$ has free variables $\{y\}$, so does ψ'_x. Hence, $\exists x.\varphi \equiv \psi_\emptyset \wedge \psi'_x \wedge \psi_y$, where the right-hand side is a hierarchical conjunction of $\mathsf{simple\text{-}FO}^{\wedge}_2$ formulas and is hence in $\mathsf{h\text{-}FO}^{\wedge}_2$. Also, observe that the signature of the right-hand side is a subset of $\{\emptyset, \{y\}\}$. This establishes the desired result.
- $\mathrm{sig}(\varphi)$ is a subset of $\{\emptyset, \{x, y\}\}$. The reasoning is entirely similar. □

First we show the reduction from $\mathsf{simple\text{-}MATLANG}$ expressions to $\mathsf{h\text{-}FO}^{\wedge}_{2,=}$ formulas. Here, the signature of the resulting formula is important to later remove the equalities.

Proposition 8.8. *Let e be a* $\mathsf{simple\text{-}MATLANG}$ *expression over $\mathcal{S}$ and let Rel be a matrix-to-relational schema encoding on $\mathcal{S}$. Let x, y be two variables such that $x \prec y$. There exists $\varphi_e \in \mathsf{h\text{-}FO}^{\wedge}_{2,=}$ simulating e such that* $\mathrm{sig}(\varphi_e)$ *is a subset of either $\{\emptyset, \{x\}, \{y\}\}$ or $\{\emptyset, \{x, y\}\}$.*

Proof. The proof is by induction and analogous to the proof of Proposition 6.5. We only illustrate two interesting cases.

- If $e = \mathbf{I}^\alpha \colon (\alpha, \alpha)$ then take $\psi_e \coloneqq x \leq \alpha \wedge y \leq \alpha \wedge x = y$. Note that $x \leq \alpha \wedge y \leq \alpha$ is a hierarchical conjunction of simple $\mathsf{FO}^{\wedge}_2$ formulas. Therefore, $\psi_e \in \mathsf{h\text{-}FO}^{\wedge}_{2,=}$, as desired.
- If $e = e_1 \times e_2$ with $e_1 \colon (1, 1)$ and $e_2 \colon (\alpha, \beta)$ we reason as follows. Let ψ_{e_1} and ψ_{e_2} be the $\mathsf{h\text{-}FO}^{\wedge}_{2,=}$ formulas simulating e_1 and e_2 obtained by induction.

  First observe that because ψ_{e_1} simulates e_1 and because e_1 always outputs scalar matrices, $[\![\psi_{e_1}]\!]_{\mathit{Rel}(\mathcal{I})}(x \mapsto i, y \mapsto j) = \mathbb{0}$ whenever $i \neq 1$ or $j \neq 1$. We conclude that therefore, $\exists x, y.\psi_{e_1}$ will always return $[\![e_1]\!](\mathcal{I})(1, 1)$ when evaluated on $\mathit{Rel}(\mathcal{I})$. Hence, we take $\psi_e \coloneqq (\exists x, y.\psi_{e_1}) \wedge \psi_{e_2}$. Note that $(\exists x, y.\psi_{e_1})$ is expressible in $\mathsf{h\text{-}FO}^{\wedge}_2$ by applying Lemma 8.7 twice, i.e., it is expressible as a hierarchical conjunction of simple $\mathsf{FO}^{\wedge}_2$ formulas. Each of these formulas must have the empty set of free variables. Therefore, their conjunction with ψ_{e_1} is also a hierarchical conjunction, and hence in $\mathsf{h\text{-}FO}^{\wedge}_{2,=}$. The signature of the resulting conjunction is the union of $\{\emptyset\}$ with the signature of ψ_{e_1}, and therefore also a subset of either $\{\emptyset, \{x\}, \{y\}\}$ or $\{\emptyset, \{x, y\}\}$. □

Next, we show how to translate any $\mathsf{qh\text{-}MATLANG}$ expression to a $\mathsf{h\text{-}FO}^{\wedge}_{2,=}$ formula.

Proposition 8.9. *Let g be a* $\mathsf{qh\text{-}MATLANG}$ *expression over $\mathcal{S}$ and let Rel be a matrix-to-relational schema encoding on $\mathcal{S}$. There exists $\psi \in \mathsf{h\text{-}FO}^{\wedge}_{2,=}$ that simulates e w.r.t. Rel.*

Proof. Let x, y be two variables such that $x \prec y$. We may assume w.l.o.g. that the formulas φ returned by Proposition 8.8 have $\mathit{free}(\varphi) = \{x, y\}$. We reason by case analysis. Let e_1, e_2 range over $\mathsf{simple\text{-}MATLANG}$ expressions.

(1) $g \in \mathsf{simple\text{-}MATLANG}$. Then the claim follows from Proposition 8.8.

(2) $g = e_1 \odot \left(e_2 \cdot \left(\mathbf{1}^\beta\right)^T\right)$ with $e_1 \colon (\alpha, \beta)$ and $e_2 \colon (\alpha, 1)$. By Proposition 8.8, there exists φ_1 and φ_2 simulating e_1 and e_2, respectively. Observe that for all matrix instances $\mathcal{I}$ and all valid indices i, j we have $[\![g]\!](\mathcal{I})_{i,j} = [\![e_1]\!](\mathcal{I})_{i,j} \odot [\![e_2]\!](\mathcal{I})_{i,1}$. Because φ_2 simulates e_2 and because e_2 always outputs a column vector, $[\![\varphi_2]\!]_{\mathit{Rel}(\mathcal{I})}(x \mapsto i, y \mapsto j) = \mathbb{0}$ whenever $j \neq 1$, for all $\mathcal{I}$. We conclude that therefore $[\![e_2]\!](\mathcal{I})_{i,1} = [\![\exists y.\varphi_2]\!]_{\mathit{Rel}(\mathcal{I})}(x \mapsto i)$. Hence it suffices take $\psi = \varphi_1(x, y) \wedge \exists y.\varphi_2(x, y)$. By Lemma 8.7 $\exists y.\varphi_2(x, y)$ is expressible in $\mathsf{h\text{-}FO}^{\wedge}_2$, i.e., it is expressible as a hiearchical conjunction of simple $\mathsf{FO}^{\wedge}_2$ formulas, which necessarily all have free variables that are a subset of $\{x\}$. Extending this conjunction

with φ_1, whose signature is a subset of $\{\emptyset, \{x\}, \{y\}\}$ or $\{\emptyset, \{x, y\}\}$ by Proposition 8.8, remains hierarchical. We conclude that ψ is expressible in $\mathsf{h}\text{-}\mathsf{FO}^{\wedge}_{2,=}$.

(3) $e = e_1 \odot (\mathbf{1}^{\alpha} \cdot e_2)$ with $e_1 \colon (\alpha, \beta)$ and $e_2 \colon (1, \beta)$. The reasoning is similar as in the previous case, but taking $\varphi = \varphi_1(x, y) \wedge \exists x.\varphi_2(x, y)$.

(4) $e = \left(e_1 \cdot \left(\mathbf{1}^{\beta}\right)^T\right) \odot (\mathbf{1}^{\alpha} \cdot e_2)$ with $e_1 \colon (\alpha, 1)$ and $e_2 \colon (1, \beta)$. The reasoning is again similar, taking $\varphi = (\exists y.\varphi_1(x, y)) \wedge (\exists x.\varphi_2(x, y))$.

(5) The only other possibilities are commutative variants of cases already considered. □

The next lemmas will aid to remove the equalities of $\mathsf{FO}^{\wedge}_{2,=}$ formulas in the final result. The reader can find their proofs in Appendix D.

Lemma 8.10. *Let φ be a $\mathsf{h}\text{-}\mathsf{FO}^{\wedge}_2$ formula (hence, without equality) using only the distinct variables x, y such that $\mathit{free}(\varphi) = \{x, y\}$. Let ψ be the $\mathsf{FO}^{\wedge}_{2,=}$ formula $\exists x.(\varphi \wedge x = y)$. Note that $\mathit{free}(\psi) = \{y\}$. There exists an $\mathsf{h}\text{-}\mathsf{FO}^{\wedge}_2$ formula ψ' equivalent to ψ.*

Lemma 8.11. *If $\varphi \in \mathsf{h}\text{-}\mathsf{FO}^{\wedge}_2$ has $\mathit{free}(\varphi) = \{x, y\}$ and $z \in \{x, y\}$ then there exists $\varphi' \in \mathsf{h}\text{-}\mathsf{FO}^{\wedge}_2$ with $\mathit{free}(\varphi') = \{z\}$ such that for all $\nu \colon \{z\}$ and all db,*

$$[\![\varphi']\!]_{db}(\nu) = [\![\varphi \wedge x = y]\!]_{db}(x \mapsto \nu(z), y \mapsto \nu(z)).$$

We are ready to show how to translate any $\mathsf{qh}\text{-}\mathsf{MATLANG}$ query to a $\mathsf{h}\text{-}\mathsf{FO}^{\wedge}_2$ query.

Corollary 8.12. *Let $\mathbf{Q}$ be an $\mathsf{qh}\text{-}\mathsf{MATLANG}$ query over $\mathcal{S}$ and let Rel be a matrix-to-relational schema encoding on $\mathcal{S}$. There exist $\mathsf{h}\text{-}\mathsf{FO}^{\wedge}_2$ query Q that simulates $\mathbf{Q}$ w.r.t. Mat.*

Proof. Let $\mathbf{H} \coloneqq e$ be a $\mathsf{qh}\text{-}\mathsf{MATLANG}$ query and let $H = \mathit{Rel}(\mathbf{H})$. We consider three cases.

(1) H is a binary relation. By Proposition 8.9 there exists $\psi \in \mathsf{h}\text{-}\mathsf{FO}^{\wedge}_{2,=}$ that simulates e w.r.t. *Rel*. By construction, ψ has two distinct free variables. Assume that x, y are these two distinct variables, and $x \prec y$. If $\psi \in \mathsf{h}\text{-}\mathsf{FO}^{\wedge}_2$ then we take Q to be the query $H(x, y) \leftarrow \psi$. Otherwise, $\psi = \psi' \wedge x = y$ with $\psi' \in \mathsf{h}\text{-}\mathsf{FO}^{\wedge}_2$ and at least one of x, y appearing free in ψ'. If $\mathit{free}(\psi') = \{x\}$ then we take $H(x, x) \leftarrow \psi'$ as simulating query. If $\mathit{free}(\psi') = \{y\}$ then we take $H(y, y) \leftarrow \psi'$ as simulating query. Otherwise $\mathit{free}(\psi') = \{x, y\}$, and we take $H(x, x) \leftarrow \psi''$ as simulating query, where ψ'' is the result of applying Lemma 8.11 to ψ' and $z = x$.

(2) H is a unary relation. Assume that $\mathbf{H}$ is of column vector type $(\alpha, 1)$; the reasoning when $\mathbf{H}$ has row vector type is similar. Because of well-typedness, also $e \colon (\alpha, 1)$. Now observe that expansions return matrices that are not column vectors (or, if they do, the result can be written without using an expansion). As such, $e \in \mathsf{simple}\text{-}\mathsf{MATLANG}$. By Proposition 8.8 there exists $\psi \in \mathsf{h}\text{-}\mathsf{FO}^{\wedge}_{2,=}$ that simulates e w.r.t. *Rel* such that $\mathrm{sig}(\psi)$ is a subset of either $\{\emptyset, \{x\}, \{y\}\}$ or of $\{\emptyset, \{x, y\}\}$. By construction, ψ has two distinct free variables. Assume that x, y are these two distinct variables, and $x \prec y$. Take $\varphi = \exists y.\psi$. Because ψ simulates e and because e always outputs column vectors, $[\![\psi]\!]_{\mathit{Rel}(\mathcal{I})}(x \mapsto i, y \mapsto j) = \mathbb{0}$ whenever $j \neq 1$, for every $\mathcal{I}$. We conclude that therefore $[\![e]\!](\mathcal{I})(i, 1) = [\![\exists y.\psi]\!]_{\mathit{Rel}(\mathcal{I})}(x \mapsto i)$ for all i and $\mathcal{I}$. As such, the query $Q' \colon H(x) \leftarrow \varphi$ simulates $\mathbf{Q}$. While Q' is a query in $\mathsf{h}\text{-}\mathsf{FO}^{\wedge}_{2,=}$, we can next apply Lemma 8.7 to φ to obtain an equivalent formula $\varphi' \equiv \varphi$ in $\mathsf{h}\text{-}\mathsf{FO}^{\wedge}_2$.

(3) H is a nullary relation. The reason is entirely similar to the previous case. □

9. Preliminaries for enumeration algorithms and query evaluation

In the previous sections we have presented a precise connection between fragments of FO^+ and fragments of sum-MATLANG. In the sequel, we move to translate the algorithmic properties of the free-connex and q-hierarchical conjunctive fragments of FO^+ to properties for the corresponding fragments of sum-MATLANG. Towards this goal, this section introduces all the extra background and definitions regarding enumeration algorithms. The properties themselves are obtained in Sections 10 and 11.

Enumeration problems. An *enumeration problem* P is a collection of pairs (I, O) where I is an *input* and O is a finite set of answers for I. There is a functional dependency from I to O: whenever (I, O) and (I', O') are in P and $I = I'$ then also $O = O'$. For this reason, we also denote O simply by $P(I)$. An *enumeration algorithm* for an enumeration problem P is an algorithm A that works in two phases: an *enumeration phase* and a *preprocessing phase*. During preprocessing, A is given input I and may compute certain data structures, but does not produce any output. During the enumeration phase, A may use the data structures constructed during preprocessing to print the elements of $P(I)$ one by one, without repetitions. The *preprocessing time* is the running time of the preprocessing phase. The *delay* is an upper bound on the time between printing any two answers in the enumeration phase; the time from the beginning of the enumeration phase until printing the first answer; and the time after printing the last answer and the time the algorithm terminates.

Query evaluation by enumeration. In our setting, we consider query evaluation as an enumeration problem. Specifically, define the *answer* of a query Q on database db to be the set of all tuples outputed by Q on db, together with their (non-zero) annotation,

$$\texttt{Answer}(Q, db) \;\coloneqq\; \{(\bar{d}, [\![Q]\!]_{db}(H)(\bar{d})) \mid [\![Q]\!]_{db}(H)(\bar{d}) \neq \mathbb{0}\}.$$

Here, H denotes the relation symbol in the head of Q.

We then associate to each FO^+ query Q over vocabulary σ and semiring $\mathcal{K}$ the following enumeration problem $\texttt{Eval}(Q, \sigma, \mathcal{K})$:

Problem:	$\texttt{Eval}(Q, \sigma, \mathcal{K})$
Input:	A $\mathcal{K}$-database db over σ
Output:	Enumerate $\texttt{Answer}(Q, db)$.

Formally this is the collection of all pairs (I, O), where $I = db$ with db a $\mathcal{K}$-database over σ; and $O = \texttt{Answer}(Q, db)$. Hence, $\texttt{Eval}(Q, \sigma, \mathcal{K})$ is the evaluation problem of Q on $\mathcal{K}$-databases.

Similarly, we associate to each sum-MATLANG query $\mathbf{Q} = \mathbf{H} := e$ over schema $\mathcal{S}$ and semiring $\mathcal{K}$ the enumeration problem $\texttt{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$:

Problem:	$\texttt{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$
Input:	A $\mathcal{K}$-instance $\mathcal{I}$ over $\mathcal{S}$
Output:	Enumerate $\texttt{Answer}(\mathbf{Q}, \mathcal{I})$.

This is, the collection of all pairs (I, O) where

- $I = \mathcal{I}$ with $\mathcal{I}$ a $\mathcal{K}$-matrix instance over $\mathcal{S}$, sparsely represented by listing for each matrix symbol $\mathbf{A}$: (α, β) the set of entries $\{(i, j, k) \mid i \leq \mathcal{I}(\alpha), j \leq \mathcal{I}(\beta), \mathbf{A}^{\mathcal{I}}_{i,j} = k \neq \mathbb{0}\}$; and
- $O = \texttt{Answer}(\mathbf{Q}, \mathcal{I})$ where, assuming $\mathbf{H}$: (α, β),

$$\texttt{Answer}(\mathbf{Q}, \mathcal{I}) \coloneqq \{(i, j, [\![\mathbf{Q}]\!](\mathcal{I})(\mathbf{H})_{i,j}) \mid [\![\mathbf{Q}]\!](\mathcal{I})(\mathbf{H})_{i,j} \neq \mathbb{0}, i \leq \mathcal{I}(\alpha), j \leq \mathcal{I}(\beta)\}.$$

Hence, $\texttt{Eval}(\mathbf{Q}, \sigma, \mathcal{K})$ is the *sparse* evaluation problem of $\mathbf{Q}$ on matrix instances over $\mathcal{S}$, where only the non-zero output entries must be computed.

Complexity classes. For two functions f and g from the natural numbers to the positive reals we define $\texttt{Enum}(f, g)$ to be the class of enumeration problems for which there exists an enumeration algorithm A such that for every input I it holds that the preprocessing time is $\mathcal{O}(f(\| I \|))$ and the delay is $\mathcal{O}(g(\| I \|))$, where $\| I \|$ denotes the size of I. For example, $\texttt{Enum}(\| db \|, 1)$ hence refers to the class of query evaluation enumeration problems that have linear time preprocessing and constant delay (in data complexity). Note that, since the inputs in an evaluation problem $\texttt{Eval}(Q, \sigma, \mathcal{K})$ consists only of the database and not the query, we hence measure complexity in *data complexity* as is standard in the literature [BDG07, IUV+20, BGS20, BKS17].

Throughout, $\| db \|$ denotes the size of the input $\mathcal{K}$-database db, measured as the length of a reasonable encoding of db, i.e., $\| db \| := \sum_{R \in \sigma} \| R^{db} \| + \sum_{c \in \sigma} 1$ where $\| R^{db} \| := (ar(R) + 1) \cdot |R^{db}|$ with $|R^{db}|$ the number of tuples $\bar{d}$ with $R^{db}(\bar{d}) \neq \mathbb{0}_{\mathcal{K}}$. Intuitively, we represent each tuple as well as its $\mathcal{K}$-value and assume that encoding a domain value or $\mathcal{K}$-value takes unit space. In addition, each constant symbol is assigned a domain value, which also takes unit space per constant symbol.

Similarly, if $\mathcal{I}$ is a matrix instance over matrix schema $\mathcal{S}$ then $\| \mathcal{I} \|$ denotes the size of a reasonable sparse encoding of I, i.e., $\| \mathcal{I} \| := \sum_{\mathbf{A} \in \mathcal{S}} \| \mathbf{A}^{\mathcal{I}} \| + \sum_{\alpha \in \sigma} 1$ where $\| \mathbf{A}^{\mathcal{I}} \|$ denotes the number of entries (i, j) with $(\mathbf{A}^{\mathcal{I}})_{i,j} \neq \mathbb{0}$, for $\mathbf{A}\colon (\alpha, \beta)$ and $1 \leq i \leq \alpha^{\mathcal{I}}$ and $1 \leq j \leq \beta^{\mathcal{I}}$.

Model of computation. As has become standard in the literature [BDG07, IUV+20, BGS20, BKS17], we consider algorithms on Random Access Machines (RAM) with uniform cost measure [AHU74]. Whenever we work with $\mathcal{K}$-databases for an arbitrary semiring $\mathcal{K}$, we assume that each semiring operation can be executed in constant time.

Complexity Hypotheses. We will use the following algorithmic problems and associated complexity hypotheses for obtaining conditional lower bounds.

(1) The *Sparse Boolean Matrix Multiplication problem*: given A and B as a list of nonzero entries, compute the nonzero entries of the matrix product AB. The *Sparse Boolean Matrix Multiplication conjecture* states that this problem cannot be solved in time $\mathcal{O}(M)$, where M is the size (number of non-zero entries) of the input plus output.
(2) The *Triangle Detection problem*: given a graph represented using adjacency lists, decide whether it contains a triangle. The *Triangle Detection conjecture* states that this problem cannot be solved in time $\mathcal{O}(M)$, where M is the size of the input plus output.
(3) $(k, k+1)$-*Hyperclique*: Given a hypergraph where every hyperedge consists of exactly $k \geq 3$ vertices, decide if it contains a hyperclique of size $k+1$. A hyperclique is a set of vertices such that each pair of vertices in the set is contained in a hyperedge. The $(k, k+1)$-*Hyperclique conjecture* states that this problem cannot be solved in time $\mathcal{O}(M)$, where M is the size of the input plus output.

Note that the three hypotheses are standard in computer science and have been used to prove conditional lower bounds for the enumeration of CQs. See [BGS20] for a discussion.

Known bounds for free-connex CQs. In what follows, let $\text{CQ}^{\not\leq}$ denote the class of CQs where no inequality atoms occur in the body. The body hence mentions only relational atoms. Also, let $\mathbb{B} = (\{\mathtt{t}, \mathtt{f}\}, \vee, \wedge, \mathtt{f}, \mathtt{t})$ be the boolean semiring.

Bagan, Durand and Grandjean [BDG07] (see also [BGS20]) proved that, under certain complexity-theoretic assumptions, the class of free-connex conjunctive queries characterizes the class of conjunctive queries that, in data complexity, can be enumerated with linear time preprocessing $\mathcal{O}(\| db \|)$ and with constant delay on boolean input databases. We next recall their results.

Theorem 9.1 [BDG07]**.** *Let Q be a $CQ^{\not\leq}$ over vocabulary σ.*

- *If Q is free-connex then $\mathtt{Eval}(Q, \sigma, \mathbb{B}) \in \mathtt{Enum}(\| db \|, 1)$.*
- *If Q is a query without self joins and $\mathtt{Eval}(Q, \sigma, \mathbb{B}) \in \mathtt{Enum}(\| db \|, 1)$, then Q is free-connex unless either the Sparse Boolean Matrix Multiplication, the Triangle Detection, or the $(k, k+1)$-Hyperclique conjecture is false.*

10. Efficient evaluation of free-connex queries

In this section, we focus on the evaluation problem for fc-MATLANG, namely, the free-connex queries in sum-MATLANG. Our aim is to transfer the known algorithmic properties of evaluating free-connex CQs to properties of evaluating fc-MATLANG.

To apply the known algorithm for fc-CQs to fc-MATLANG, we must first solve two problems: (1) the efficient evaluation algorithm for fc-CQs is only established for evaluation over Boolean semiring, not arbitrary semirings $\mathcal{K}$, and (2) the algorithm is for CQ without inequalities (comparison atoms). We will therefore generalize the known evaluation algorithms to other semirings (e.g., $\mathbb{R}$), and to queries that have also inequalities (Section 10.1). Similarly, we do the same for the lower bounds known for fc-CQs (Section 10.2).

10.1. Upper bounds for free-connex queries.

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* [Gol13]. Note that semirings used in practice, like $\mathbb{B}$, $\mathbb{N}$, and $\mathbb{R}$, are semi-integral domains. We will establish:

Theorem 10.1. *Let $\mathcal{K}$ be a semi-integral domain. For every free-connex query Q over σ, $\mathtt{Eval}(Q, \sigma, \mathcal{K})$ can be evaluated with linear-time preprocessing and constant-delay. In particular, $\mathtt{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ 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 can only be produced by some zero input 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 $a \odot b = \mathbb{0}$. Consider $Q : H(x, y) \leftarrow R(x) \wedge S(y)$ and a database db over the previous semiring such that $R(1) = a; R(2) = a; S(1) = b$ and $S(2) = b$. Then, the output of $\mathtt{Eval}(Q, \sigma, \mathcal{K})$ with input db is empty, although the body can be instantiated in four different ways, all of them producing $\mathbb{0}$ values.

The upper bounds for the similar yet different setting presented in [ECK24] are with respect to direct-access. In this context, Theorem 10.1 straightforwardly yields linear time preprocessing and linear time direct-access, in contrast to the loglinear time preprocessing and logarithmic time direct-access achieved in [ECK24]. However, in this work, tuples with zero-annotated values will not be part of the query result.

To illustrate the utility of Theorem 10.1, 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 10.1 tells us that we may evaluate the query in time $\mathcal{O}(\| \mathbf{A} \| + \| \mathbf{U} \| + \| \mathbf{V} \|)$.

We derive the enumeration algorithm for $\texttt{Eval}(Q, \sigma, \mathcal{K})$ by extending the algorithm of [BDG07] to any semi-integral domain $\mathcal{K}$ and taking care of the inequalities in Q. Then, 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 6.2.

We now prove Theorem 10.1 in two steps. We first transfer the upper bound of CQs$^{\not\leq}$ over $\mathbb{B}$-databases to CQs$^{\not\leq}$ over $\mathcal{K}$-databases. We then generalize the latter to CQs over $\mathcal{K}$-databases.

Remark. In what follows, we will describe enumeration algorithms for $\texttt{Eval}(Q, \sigma, \mathcal{K})$. Formally, given an input database db, these algorithms hence have to enumerate the set $\texttt{Answer}(Q, db)$ of output tuples with their non-zero semiring annotation. In the proofs however, we will often find it convenient to instead enumerate the set $\texttt{Vals}(Q, db)$ of valuations,

$$\texttt{Vals}(Q, db) \coloneqq \{(\nu, [\![Q]\!]_{db}(\nu)) \mid \nu \colon \overline{x}, [\![Q]\!]_{db}(\nu) \neq \mathbb{0}\}.$$

Note that an algorithm that enumerates $\texttt{Vals}(Q, db)$ can also be used to enumerate $\texttt{Answer}(Q, db)$ by turning each valuation into an output tuple, and vice versa an algorithm that enumerates $\texttt{Answer}(Q, db)$ can also be used to enumerate $\texttt{Vals}(Q, db)$: by definition every output tuple $\overline{d} \models \overline{x}$ such that $[\![Q]\!]_{db}(H)(\overline{d}) \neq \mathbb{0}$ induces a valuation $\nu \colon \overline{x}$ such that $[\![Q]\!]_{db}(\nu) \neq \mathbb{0}$. Note in particular that if either one of them can be enumerated with constant delay, then so can the other. We use this insight repeatedly in the proofs.

From $\mathbb{B}$-databases to $\mathcal{K}$-databases over a semi-integral domain. We start our proof of Theorem 10.1 by extending the algorithmic results of free connex CQ from the boolean semiring to any semi-integral domain.

Proposition 10.2. *If Q is a free-connex CQ$^{\not\leq}$ over σ and $\mathcal{K}$ a semi-integral domain then $\texttt{Eval}(Q, \sigma, \mathcal{K}) \in \texttt{Enum}(\| db \|, 1)$.*

Proof. Because Q is free-connex, it has a query plan (T, N) by Proposition 5.4. We describe different evaluation algorithms depending on the shape of Q and (T, N). Let db be a $\mathcal{K}$-database over σ and assume the semi-integral domain $\mathcal{K} = (K, \oplus, \odot, \mathbb{0}, \mathbb{1})$.

(1) When Q is full, i.e, no variable is quantified, we reduce to the Boolean case as follows. Assume that $Q = H(\overline{z}) \leftarrow R_1(\overline{z}_1) \wedge \cdots \wedge R_n(\overline{z}_n)$. In $\mathcal{O}(\| db \|)$ time construct $\texttt{Bool}(db)$, the $\mathbb{B}$-database obtained from db defined as:

$$R^{\texttt{Bool}(db)} \colon \overline{d} \mapsto \begin{cases} \texttt{t} \text{ if } R^{db}(\overline{d}) \neq \mathbb{0} \\ \texttt{f} \text{ otherwise,} \end{cases}$$

for every $R \in \sigma$. Furthermore, preprocess db by creating lookup tables such that for every relation R and tuple $\overline{a}$ of the correct arity we can retrieve the annotation $R^{db}(\overline{a})$ in time $\mathcal{O}(|\overline{a}|)$ time. Note that since σ is fixed, $|\overline{a}|$ is constant. This retrieval is hence $\mathcal{O}(1)$ in data complexity. It is well-known that such lookup tables can be created in $\mathcal{O}(\| db \|)$ time in the RAM model.

Finally, invoke the algorithm for $\texttt{Eval}(Q, \sigma, \mathbb{B})$ on $\texttt{Bool}(db)$. By Theorem 9.1, this algorithm has $\mathcal{O}(\| \texttt{Bool}(db) \|) = \mathcal{O}(\| db \|)$ preprocessing time, after which we may

enumerate $\texttt{Answer}(Q, \texttt{Bool}(db))$ with $\mathcal{O}(1)$ delay. We use the latter to enumerate $\texttt{Answer}(Q, db)$ with $\mathcal{O}(1)$ delay: whenever an element $(\overline{a}, \texttt{t})$ of $\texttt{Answer}(Q, \texttt{Bool}(db))$ is enumerated retrieve $k_1 := R_1(\overline{a}|_{\overline{z}_1}), \ldots, k_n := R_n(\overline{a}|_{\overline{z}_n})$ by means of the previously constructed lookup tables. Here, $\overline{a}|_{\overline{z}_n}$ denotes the projection of $\overline{a}$ according to the variables in $\overline{z}_i$. Then output $(\overline{a}, k_1 \odot \ldots \odot k_n)$. Because the query is fixed, so is the number of lookups we need to do, and this hence takes $\mathcal{O}(1)$ additional delay in data complexity. In summary the delay between printing one output and the next is hence $\mathcal{O}(1)$ while the total processing time—comprising the construction of $\texttt{Bool}(db)$, the lookup tables and the preprocessing phase of $\texttt{Eval}(Q, \sigma, \texttt{Bool})$— is $\mathcal{O}(\| db \|)$.

We claim that the set of pairs $(\overline{a}, k_1 \odot \ldots \odot k_n)$ hence printed is exactly $\texttt{Answer}(Q, db)$:

- If we print $(\overline{a}, k_1 \odot \ldots \odot k_n)$ then $(\overline{a}, \texttt{t})$ was in $\texttt{Answer}(Q, \texttt{Bool}(db))$, which can only happen if $R_i^{\texttt{Bool}(db)}(\overline{a}|_{\overline{z}_i}) = \texttt{t}$ for every $1 \leq i \leq n$. By definition of $\texttt{Bool}(db)$ this means that $k_i = R_i^{db}(\overline{a}|_{\overline{z}_i}) \neq \mathbb{0}$. Hence, $[\![Q]\!]_{db}(H)(\overline{a}) = k_1 \odot \ldots \odot k_n$ since Q is full. Because $\mathcal{K}$ is a semi-integral domain, the latter product is non-zero and hence $(\overline{a}, k_1 \odot \ldots \odot k_n) \in \texttt{Answer}(Q, db)$.
- If $(\overline{a}, k) \in \texttt{Answer}(Q, db)$ then by definition $k \neq \mathbb{0}$ and, because Q is full, $k = k_1 \odot \ldots \odot k_n$ where $k_i = R_i^{db}(\overline{a}|_{\overline{z}_i})$. As such $k_i \neq \mathbb{0}$ for $1 \leq i \leq n$. Hence, $R_i^{\texttt{Bool}(db)}(\overline{a}|_{\overline{z}_i}) = \texttt{t}$ for every i and thus, $(\overline{a}, \texttt{t}) \in \texttt{Answer}(Q, \texttt{Bool}(db))$. Therefore, $(\overline{a}, k_1 \odot \ldots \odot k_n)$ will be enumerated by our algorithm above.

(2) When the query plan (T, N) is such that N consists of a single node $N = \{r\}$ we next show that may compute the entire result of Q in $\mathcal{O}(\| db \|)$ time. This implies that $\texttt{Eval}(Q, \sigma, \mathcal{K}) \in \texttt{Enum}(\| db \|, 1)$: compute $\texttt{Answer}(Q, db)$ and use this data structure (e.g., represented as a linked list) to enumerate its elements, which trivially supports constant delay. Essentially, the argument that we use here is the same as the standard argument in the theory of acyclic conjunctive queries that Boolean acyclic queries on Boolean databases can be evaluated in linear time by a series of projection operations and semijoin operations [Yan81, AHV95]. We repeat the argument in the $\mathcal{K}$-setting here for completeness only. For notational convenience, if ψ is an $\mathsf{FO}^\wedge$ formula then we denote by $\texttt{Answer}(\psi, db)$ the set $\texttt{Answer}(P_\psi, db)$ where $P_\psi : H(\overline{x}) \leftarrow \psi$ with $\overline{x}$ a repetition-free list of the free variables of ψ. The algorithm exploits the fact that, since (T, N) is a query plan for Q we have the body of Q is equivalent to $\varphi[T, N] = \varphi[T, r]$. Hence, it suffices to show that $\texttt{Answer}(\varphi[T, r], db)$ may be computed in linear time.[3]

We argue inductively on the height of T that $\texttt{Answer}(\varphi[T, N], db)$ may be computed in linear time. By Lemma 5.5, we may assume without loss of generality that for every node n with children c_1, c_2 in T either $var(c_1) = var(n)$ and $var(c_2) \subseteq var(n)$; or $var(c_2) = var(n)$ and $var(c_1) \subseteq var(n)$.

(a) When T has height 1 then the root r is a leaf and hence an atom, say $R(\overline{x})$. As such, $\texttt{Answer}(\varphi[T, r], db)$ can trivially be computed in $\mathcal{O}(\| db \|)$ time by scanning R^{db}.

(b) When T has height greater than 1 we identify two cases.
- Root r has one child c. By Lemma 5.8 we have $\varphi[T, r] \equiv \exists \overline{y}.\varphi[T, c]$ with $\overline{y} = var(c) \setminus var(r)$. Here, $\varphi[T, c]$ equals $\varphi[T_c, c]$ with T_c the subtree of T rooted at c. As such, we may compute $\texttt{Answer}(\varphi[T, r], db)$ by first computing $A := \texttt{Answer}(\varphi[T_c, c], db)$ and then projecting on the variables in $var(r)$. This is in

[3]The only difference between Q and $\varphi[T, r]$ is that Q may repeat some variables in the head; the result of Q can hence be obtained by computing $\varphi[T, r]$ and repeating values in each result tuple as desired.

linear time by induction hypothesis and the fact that projections on $\mathcal{K}$-relations can be done in linear time in the RAM model, even on general $\mathcal{K}$-databases.[4]

- Root r has two children c_1 and c_2. We may assume w.l.o.g. that $var(r) = var(c_1)$ and $var(c_2) \subseteq var(r) = var(c_1)$. By Lemma 5.9 we have $\varphi[T, r] \equiv \varphi[T, c_1] \wedge \varphi[T, c_2]$. Note that because $var(c_1) \subseteq var(c_2)$ and $free(\varphi[T, c_i]) = var(c_i)$, the formula $\varphi[T, r]$ is hence expressing a semijoin between $\varphi[T, c_1]$ and $\varphi[T, c_2]$. Hence, we may compute $\texttt{Answer}(\varphi[T, r],)$ by first computing $A_1 := \texttt{Answer}(\varphi[T_1, c_1], db)$ and $A_2 := \texttt{Answer}(\varphi[T_2, c_2], db)$ (where T_1 and T_2 are the subtrees of T rooted at c_1, resp. c_2) and then computing the semijoin between A_1 and A_2. This is in linear time by induction hypothesis and the fact that semijoins on $\mathcal{K}$-relations can be done in linear time in the RAM model, even on general $\mathcal{K}$-databases.[5]

(3) When Q is not full and N has more than one node we reduce to the previous two cases as follows. By Lemma 5.7 we have that $\varphi[T, N] \equiv \varphi[T_1, n_1] \wedge \cdots \wedge \varphi[T_l, n_l]$ where $F = \{n_1, \ldots, n_l\}$ is the frontier of N, i.e, the nodes without children in N, and $T_1, \ldots, T_l$ are the subtrees of T rooted at $n_1, \ldots, n_l$ respectively. Hence, we may compute $\texttt{Answer}(\varphi[T, N], db)$ by computing $A_i := \texttt{Answer}(\varphi[T_i, n_i], db)$ for $1 \leq i \leq n$ and then taking the join of the resulting $\mathcal{K}$-relations. By item (2) above, each $\texttt{Answer}(\varphi[T_i, n_i], db)$ can be computed in linear time, and is hence of linear size. The preprocessing and enumeration of the final join can be done as in item (1) above, since this is a full join. □

From CQ$^{\not\leq}$ to CQ. An inequality atom $x \leq c$ is *covered* in a CQ Q if there is a relational atom in Q that mentions x; it is *non-covered* otherwise. So, in the query $H(y, z) \leftarrow R(x, y) \wedge y \leq c \wedge z \leq d$ the inequality $y \leq c$ is covered but $z \leq d$ is not. For a CQ Q define the *split* of Q to be the pair $\texttt{split}(Q) = (Q_{\text{rel}}, Q_{\text{ineq}})$ where:

$$Q_{\text{rel}}(\overline{x}) \leftarrow \exists \overline{y}. \bigwedge_{R(\overline{z}) \in at(\varphi)} R(\overline{z}) \qquad Q_{\text{ineq}}(\overline{u}) \leftarrow \exists \overline{v}. \bigwedge_{w \leq c \in at(\varphi), \text{non-covered}} w \leq c$$

with $\overline{x}$ and $\overline{y}$ the set of all free resp. bound variables in Q that occur in a relational atom in Q, and $\overline{u}$ and $\overline{v}$ the set of free resp. bound variables in Q that occur in a non-covered inequality in Q. In what follows, we call Q_{rel} the relational part of Q.

We note that if Q does not have any non-covered inequality then Q_{ineq} is in principle the empty query. For convenience, we then set it equal to $Q_{\text{ineq}}() \leftarrow \exists x. x \leq 1$ in that case, which is equivalent to $\texttt{t}$.

The next proposition relates free-connexness of a CQ with inequalities to the free-connexness of its relational part.

Proposition 10.3. *Let Q be a CQ and assume $\texttt{split}(Q) = (Q_{rel}, Q_{ineq})$. Q is free-connex if, and only if, Q_{rel} is free-connex.*

[4]Create an empty lookup table. Iterate over the tuples $(\overline{a}, k)$ of A. For every such tuple $(\overline{a}, k)$, look up $\overline{a}|_{var(r)}$ in the lookup table. If this is present with $\mathcal{K}$-value l, then set $l := l \oplus k$; otherwise add $(\overline{a}|_{var(r)}, k)$ to the table. After the iteration on A, enumerate the entries $(\overline{b}, k)$ in the lookup table and add each entry to the output when $k \neq \mathbb{0}$. This is exactly the projected result.

[5]Create a lookup table on A_2, allowing to retrieve the $\mathcal{K}$-annotation given a $var(c_2)$-tuple. Then iterate over the entries $(\overline{a}, k)$ of A_1. For every such tuple $(\overline{a}, k)$, look up $\overline{a}|_{var(c_2)}$ in the lookup table. If this is present with $\mathcal{K}$-value l, then output $(\overline{a}, k \odot l)$ provided $k \odot l \neq \mathbb{0}$.

Proof. This is essentially because unary atoms (like inequality atoms) can never affect the (a)cyclicity of a CQ: if a CQ is acyclic it remains so if we add or remove unary atoms from its body. □

The following property relates the evaluation of a CQ (with inequalities) to the evaluation of its relational part. Let Q be a CQ over vocabulary σ. To each relational atom $\alpha = R(\overline{x})$ we associate the query

$$Q[\alpha] : H(\overline{x}) \leftarrow R(\overline{x}) \wedge \iota_1 \wedge \cdots \wedge \iota_\ell$$

where $\iota_1, \ldots, \iota_\ell$ are all inequality atoms in Q covered by α. We say that a database db is *atomically consistent* w.r.t Q if for every relation symbol $R \in \sigma$ of arity n, every atom $\alpha = R(\overline{x})$ in Q over relation symbol R, and every n-tuple $\overline{a}$ we have

$$\text{if } R^{db}(\overline{a}) \neq \mathbb{0} \text{ and } \overline{a} \models \overline{x} \qquad \text{then} \qquad [\![Q[\alpha]]\!]_{db}(H)(\overline{a}) \neq \mathbb{0}.$$

Intuitively, if db is atomically consistent, then every tuple in a relation R that satisfies the atom $\alpha = R(\overline{x})$[6] also satisfies all inequalities covered by α. Note that we may always make a given db atomically consistent with Q in $\mathcal{O}(\| db \|)$ time: simply scan each relation R of db and for each tuple $\overline{a}$ loop over the R-atoms $\alpha = R(\overline{x})$ in Q. If $\overline{a} \models \overline{x}$ then check that the constraints imposed by $Q[\alpha]$ are satisfied. Since $Q[\alpha]$ is a full query, this can be done in constant time per tuple in data complexity. We call this process *atomically reducing* db, that is, of making db atomically-consistent with Q .

The following claims are straighforward to verify.

Claim 10.4. Let Q be a CQ over vocabulary σ, let db be a database and let db' be its atomic reduction. Then $\texttt{Answer}(Q, db) = \texttt{Answer}(Q, db')$.

Claim 10.5. Let $Q(\overline{x})$ be a CQ with $\texttt{split}(Q) = (Q_{\text{rel}}(\overline{x}_1), Q_{\text{ineq}}(\overline{x}_2))$. If db is atomically consistent with Q then $[\![Q]\!]_{db}(\nu) = [\![Q_{\text{rel}}]\!]_{db}(\nu|_{\overline{x}_1}) \odot [\![Q_{\text{ineq}}]\!]_{db}(\nu|_{\overline{x}_2})$ for every valuation $\nu \colon \overline{x}$.

Lemma 10.6. *Let Q be a CQ over vocabulary σ with $\texttt{split}(Q) = (Q_{rel}, Q_{ineq})$ and let $\mathcal{K}$ be a semiring (not necessarily a semi-integral domain). Then $\texttt{Eval}(Q_{ineq}, \sigma, \mathcal{K}) \in \texttt{Enum}(\| db \|, 1)$.*

Proof. Let $x_1, \ldots, x_n$ be the free variables of Q_{ineq} and $y_1, \ldots, y_m$ its quantified variables. Let $\#z$ denote the number of inequalities mentioning variable z (free or quantified). Assume w.l.o.g. that Q_{ineq} is of the form

$$H(x_1, \ldots, x_n) \leftarrow \exists y_1, \ldots, y_m. \bigwedge_{1 \leq i \leq n} \bigwedge_{j \leq \# x_i} (x_i \leq c_j^i) \wedge \bigwedge_{1 \leq p \leq m} \bigwedge_{q \leq \# y_j} (y_p \leq d_q^p).$$

Let db be a $\mathcal{K}$-database over σ. It is straightforward to verify that $((a_1, \ldots, a_n), k) \in \texttt{Answer}(Q_{\text{ineq}}, db)$ if, and only if,

- For every $1 \leq i \leq n$ we have that $a_i \leq c_i$ where $c_i := \min_{1 \leq j \leq \# x_i} db(c_j^i)$;
- $k \neq \mathbb{0}$; and
- $k = \underbrace{(\mathbb{1} \odot \ldots \odot \mathbb{1})}_{\#y_1 \text{ times}} \oplus \ldots \oplus \underbrace{(\mathbb{1} \odot \ldots \odot \mathbb{1})}_{\#y_m \text{ times}} = \underbrace{\mathbb{1} \oplus \ldots \oplus \mathbb{1}}_{m \text{ times}}$

In particular, the annotation k is the same for each output tuple. Because m depends only on the query, we may compute k in constant time. Furthermore, we may compute the numbers $c_i \in \mathbb{N}$ in $\mathcal{O}(\| db \|)$ time. Computing c_i and k constitutes the preprocessing step of our enumeration algorithm. Having these numbers computed, it is then straightforward

[6]Note that $\overline{x}$ may repeat variables, so an atom like $S(x, y, x)$ requires the first and third components of $\overline{a}$ to be equal.

to enumerate the tuples: indicate end of enumeration immediately if $k = \mathbb{0}$; otherwise enumerate all tuples $((a_1, \ldots, a_n), k) \in \texttt{Answer}(Q_{\text{ineq}}, db)$ with constant delay by means of a nested loop. □

Proposition 10.7. *Let Q be a CQ over σ with $\texttt{split}(Q) = (Q_{rel}, Q_{ineq})$. If it holds that $\texttt{Eval}(Q_{rel}, \sigma, \mathcal{K}) \in \texttt{Enum}(\| db \|, 1)$ and $\mathcal{K}$ is a semi-integral domain then $\texttt{Eval}(Q, \sigma, \mathcal{K}) \in \texttt{Enum}(\| db \|, 1)$.*

Proof. Let db be the input database. First make db atomically consistent with Q, which takes time $\mathcal{O}(\| db \|)$. Let db' be the resulting database. Note that, by Claim 10.4, we have $\texttt{Vals}(Q, db) = \texttt{Vals}(Q, db')$. Because $\| db' \| = \mathcal{O}(\| db \|)$ it hence suffices to show that we may enumerate $\texttt{Vals}(Q, db')$ with constant delay after linear time preprocessing.

Thereto, we reason as follows. Because $\texttt{Eval}(Q_{\text{rel}}, \sigma, \mathcal{K}) \in \texttt{Enum}(\| db \|, 1)$ we may compute a data structure in $\mathcal{O}(\| db' \|)$ time that allows us to enumerate $\texttt{Vals}(Q_{\text{rel}}, db')$ with constant delay. Furthermore, by Lemma 10.6 we may compute in $\mathcal{O}(\| db' \|)$ time another data structure that allows us to enumerate $\texttt{Vals}(Q_{\text{ineq}}, db')$ with constant delay. Computing both data structures constitutes our preprocessing step. By Claim 10.5 we can then use both data structures to enumerate $\texttt{Vals}(Q, db')$ with constant delay as follows:

- For each $(\nu_1, k_1) \in \texttt{Vals}(Q_{\text{rel}}, db^Q_{\text{rel}})$
  - For each $(\nu_2, k_2) \in \texttt{Vals}(Q_{\text{ineq}}, db)$, output $(\nu_1 \cup \nu_2, k_1 \odot k_2)$.

Note that $k_1 \odot k_2 \neq \mathbb{0}$ as required since $k_1 \neq \mathbb{0}$, $k_2 \neq \mathbb{0}$ and $\mathcal{K}$ is a semi-integral domain. □

We can finally prove Theorem 10.1.

Proof of Theorem 10.1. If $Q = (Q_{\text{rel}}, Q_{\text{ineq}})$ is free-connex then by Proposition 10.3 we have that Q_{rel} is free-connex. By Proposition 10.2 and because $\mathcal{K}$ is a semi-integral domain, $\texttt{Eval}(Q_{\text{rel}}, \sigma, \mathcal{K}) \in \texttt{Enum}(\| db \|, 1)$. And finally due to Proposition 10.7, the latter implies $\texttt{Eval}(Q, \sigma, \mathcal{K}) \in \texttt{Enum}(\| db \|, 1)$.

Now we prove the second part of the result, i.e., achieve the same processing complexity for the matrix evaluation problem $\texttt{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$. Fix a matrix-to-relational schema encoding *Rel* on $\mathcal{S}(\mathbf{Q})$ such that the head atom of Q is binary. The former requirement arises purely due to $\texttt{Answer}(\mathbf{Q}, \mathcal{I})$ being defined as $\{(i, j, [\![\mathbf{Q}]\!](\mathcal{I})(\mathbf{H})_{i,j}) \mid [\![\mathbf{Q}]\!](\mathcal{I})(\mathbf{H})_{i,j} \neq \mathbb{0}, i \leq \mathcal{I}(\alpha), j \leq \mathcal{I}(\beta)\}$ where $\mathbf{Q} = \mathbf{H} := e$ and $\mathbf{H}\colon (\alpha, \beta)$.

By Corollary 6.2 there exists a free-connex CQ Q over vocabulary $\sigma(Q) = \mathit{Rel}(\mathcal{S}(\mathbf{Q}))$ that simulates $\mathbf{Q}$ under *Rel*. As such, we may reduce $\texttt{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ to $\texttt{Eval}(Q, \sigma, \mathcal{K})$: given a matrix instance $\mathcal{I}$ over $\mathcal{S}$, compute $\mathit{Rel}(\mathcal{I})$ and call the enumeration algorithm for $\texttt{Eval}(Q, \sigma, \mathcal{K})$. By definition of simulation, the set $\texttt{Answer}(Q, \mathit{Rel}(\mathcal{I}))$ is exactly the same as $\texttt{Answer}(\mathbf{Q}, \mathcal{I})$. Hence, because $\texttt{Eval}(Q, \sigma, \mathcal{K}) \in \texttt{Enum}(\| db \|, 1)$ and because $\| \mathit{Rel}(\mathcal{I}) \| = \mathcal{O}(\| \mathcal{I} \|)$ it follows that $\texttt{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K}) \in \texttt{Enum}(\| \mathcal{I} \|, 1)$. □

10.2. **Lower bounds for free-connex queries.** Next, we show how to extend the lower bounds of [BDG07] and [BGS20] to our setting. As in those works, our lower bounds are for CQ *without self-joins*. Further, we need some additional restrictions for inequalities. Recall that an inequality $x \leq c$ in Q is *covered*, if there exists a relational atom in Q that mentions 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* [HW96, HW98] if for all $a, b \in K$ it holds that $a \oplus b = \mathbb{0}$ implies $a = b = \mathbb{0}$. The *subsemiring of* $\mathcal{K}$ *generated* by $\mathbb{0}$ and $\mathbb{1}$ is the semiring $\mathcal{K}' = (K', \oplus, \odot, \mathbb{0}, \mathbb{1})$ where K' is the smallest set closed that (i) contains $\mathbb{0}$ and $\mathbb{1}$ and (ii) is closed under $\oplus$ and $\odot$.

We will extend the lower bound from [BDG07, BGS20] as follows.

Theorem 10.8. *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 $\texttt{Eval}(Q, \sigma, \mathcal{K})$ 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. In contrast, lower bounds in [ECK24] require the semiring to be *idempotent*, meaning that $a \oplus a = a$ for all $a \in K$.

To see why the constant-disjointness condition is necessary, consider the CQ:

$$Q : H(x, z) \leftarrow \exists y.R(x, y) \wedge S(y, z) \wedge y \leq 1.$$

This query violates condition (i) of constant-disjointness, and is not free-connex. Nevertheless, its evaluation is in $\texttt{Enum}(\| db \|, 1)$: because y can take only a single value, we can evaluate Q by doing preprocessing and enumeration for Q': $H(x, y, z) \leftarrow R(x, y) \wedge S(y, z) \wedge y \leq 1$ instead, which is free-connex, hence in $\texttt{Enum}(\| db \|, 1)$. Whenever a tuple (x, y, z) is enumerated for Q' we yield the tuple (x, z) for the enumeration of Q. This is also constant-delay and without duplicates due to the inequality $y \leq 1$.

A similar argument holds for condition (ii) of constant-disjointness. Consider

$$P\colon H(x, z, u) \leftarrow \exists y.R(x, y) \wedge S(y, z) \wedge y \leq c \wedge u \leq c,$$

which violates condition (ii) of constant-disjointess. Again, this query is not free-connex but its evaluation is in $\texttt{Enum}(\| db \|, 1)$. To see why, consider the query P': $H(x, y, z) \leftarrow R(x, y) \wedge S(y, z) \wedge y \leq c$, which is free-connex and therefore in $\texttt{Enum}(\| db \|, 1)$. To evaluate P we simply do all the (linear-time) preprocessing for P' and use P' enumeration procedure to enumerate P with constant delay. This works as follows. During the enumeration of P maintain a lookup table L mapping (x, z) tuples to the smallest natural number u such that all tuples (x, z, u) that have already been output for P satisfy $u \leq L(x, z)$. If $L(x, z) = 0$ then no entry occurs in the lookup table. To enumerate P we invoke P' enumeration procedure. When a tuple (x, y, z) is enumerated by P', we output $(x, z, L(x, z) + 1)$ for the enumeration of P. Additionally, we update $L(x, z) := L(x, z) + 1$. Once P''s enumeration is exhausted, we iterate over the entries of the lookup table, and for each such entry (x, z) output all remaining tuples (x, z, u) with $L(x, z) < u' \leq c$. This last step is not necessarily constant-delay (there may be an unbounded number of entries with $L(x, z) = c$, yielding no output), but this can be fixed by removing (x, z) from L during the enumeration of P' whenever $L(x, z)$ is updated to become c. The constraint that $y \leq c \wedge u \leq c$ ensures that the enumeration is still correct.

Theorem 10.8 allows to also derive lower bounds for conj-MATLANG. 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 5.1. In order to achieve this, the prenex normal form of conj-MATLANG expressions (defined in Section 4) proves useful. Let $e = \Sigma\overline{\mathbf{v}}.\ \mathbf{s}_1 \times \cdots \times \mathbf{s}_n \times \mathbf{x} \cdot \mathbf{y}^T$ be a conj-MATLANG sentence in prenex normal form. A vector variable $\mathbf{v} \in \overline{\mathbf{v}}$ is *covered* in e if (i) there exists a scalar subexpression $\mathbf{s}_i$ of the form $\mathbf{v}^T \cdot \mathbf{A} \cdot \mathbf{u}$ or $\mathbf{u}^T \cdot \mathbf{A} \cdot \mathbf{v}$; or (ii) if there exists scalar subexpression $\mathbf{s}_i$ doing a vector multiplication $\mathbf{v}^T\mathbf{u}$ or $\mathbf{u}^T\mathbf{v}$ with $\mathbf{u}$ covered in e. (Note that the latter notion is recursive.) The sentence e is *constant-disjoint* if for every pair $\mathbf{v}, \mathbf{w} \in \overline{\mathbf{v}}$ of quantified vector variables of the same type, $\mathbf{v}\colon (\gamma, 1)$ and $\mathbf{w}\colon (\gamma, 1)$, if $\mathbf{v}$ is covered and $\mathbf{w}$ is not then $\mathbf{w} \notin \{\mathbf{x}, \mathbf{y}\}$. A conj-MATLANG sentence $e\colon (\alpha, \beta)$ is *constant-disjoint* if its conversion into prenex normal form is constant-disjoint.

The former is straightforwardly extended to a conj-MATLANG query $\mathbf{Q}$. The following lower bound is now attainable.

Corollary 10.9. *Let $\mathbf{Q}$ be a* conj-MATLANG *query over $\mathcal{S}$ such that $\mathbf{Q}$ does not repeat matrix symbols and $\mathbf{Q}$ is 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 $\mathit{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ 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 remainder of this section is devoted to proving Theorem 10.8 and Corollary 10.9.

Lower bound for free-connex CQs. We begin by proving Theorem 10.8, which is done in two steps. First we show that constant-disjointness is a sufficient condition for us to be able to reduce the evaluation problem of CQs$^{\not\leq}$ over $\mathbb{B}$-databases to the evaluation problem of CQs over $\mathbb{B}$-databases. Second, we show that the evaluation problem of CQs over $\mathbb{B}$-databases can be reduced to the evaluation problem of CQs over $\mathcal{K}$-databases, assuming that the subsemiring generated by $\mathbb{0}_\mathcal{K}$ and $\mathbb{1}_\mathcal{K}$ is zero-sum free.

Proposition 10.10. *Let Q be a constant-disjoint CQ over σ with $\mathit{split}(Q) = (Q_{rel}, Q_{ineq})$. If Q is constant-disjoint and $\mathit{Eval}(Q, \sigma, \mathbb{B}) \in \mathit{Enum}(\| db \|, 1)$ then $\mathit{Eval}(Q_{rel}, \sigma, \mathbb{B}) \in \mathit{Enum}(\| db \|, 1)$.*

Proof. We reduce $\texttt{Eval}(Q_{\text{rel}}, \sigma_{\text{rel}}, \mathbb{B})$ to $\texttt{Eval}(Q, \sigma, \mathbb{B})$. Denote the free variables of Q, Q_{rel} and Q_{ineq} by $\overline{x}$, $\overline{x}_1$ and $\overline{x}_2$, respectively. Note that $\overline{x}_1$ and $\overline{x}_2$ are disjoint and $\mathit{var}(\overline{x}) = \mathit{var}(\overline{x}_1) \cup \mathit{var}(\overline{x}_2)$. Consider an arbitrary database db_{rel} over σ, input to $\texttt{Eval}(Q_{\text{rel}}, \sigma, \mathbb{B})$. Construct the database db, input to $\texttt{Eval}(Q, \sigma, \mathbb{B})$ as follows.

Recall that all data values occurring in tuples in db_{rel} are non-zero natural numbers. Scan db_{rel} and compute the largest natural number M occurring in db_{rel}. Then construct the database db by setting

- $R^{db} = R^{db_{\text{rel}}}$ for each relation symbol $R \in \sigma$;
- $db(c) := M$ for all constant symbols $c \in \sigma$ that occur in a covered inequality in Q. Note that $c \neq 1$ because Q is constant-disjoint, and hence we are allowed to map $c \mapsto M$;
- $db(c) := 1$ for all other constant symbols $c \in \sigma$.

The computation of db is clearly in $\mathcal{O}(\| db_{\text{rel}} \|)$ and $\| db \| = \| db_{\text{rel}} \|$.

Note that, because Q_{rel} consists of relational atoms only, its evaluation is hence independent of the values assigned to the constant symbols by db_{rel}. Therefore, $\texttt{Answer}(Q_{\text{rel}}, db_{\text{rel}}) = \texttt{Answer}(Q_{\text{rel}}, db)$. To obtain the proposition, it hence suffices to show that $\texttt{Answer}(Q_{\text{rel}}, db)$ may be enumerated with constant delay after further linear time preprocessing. We do so by showing that $\texttt{Vals}(Q_{\text{rel}}, db,)$ may be enumerated with constant delay after linear

time preprocessing as follows. Observe that db is atomically consistent with Q. This is by construction since for every covered inequality $x \leq c$ we have $db(c)$ equal to M, with M larger than any value in db. Hence, by Claim 10.5 we have

$$\texttt{Vals}(Q, db) = \{(\nu_1 \cup \nu_2, \texttt{t}) \mid (\nu_1, \texttt{t}) \in \texttt{Vals}(Q_{\text{rel}}, db), (\nu_2, \texttt{t}) \in \texttt{Vals}(Q_{\text{ineq}}, db)\}.$$

As such, $\texttt{Vals}(Q_{\text{rel}}, db) = \{(\nu|_{\overline{x}_1}, \texttt{t}) \mid (\nu, \texttt{t}) \in \texttt{Vals}(Q, db)\}$ and we may hence enumerate $\texttt{Vals}(Q_{\text{rel}}, db)$ by enumerating the elements of $\texttt{Vals}(Q, db)$ and for each such element output $(\nu|_{\overline{x}_1}, \texttt{t})$. Note that the resulting enumeration is necessarily duplicate-free as required: by definition of db the set $\texttt{Vals}(Q_{\text{ineq}}, db)$ consists of a single element: the pair $(\overline{x}_2 \mapsto 1, \texttt{t})$. To see why this is neccesarily the case observe that, by definition, all free variables $y \in \overline{x}_2$ of Q_{ineq} must occur in an non-covered inequality $y \leq c$. Because Q is constant-disjoint this constant c is such that there is no covered inequality $z \leq d$ in Q with $c = d$. (If there were, y would be non-free.) This means that we have set $c^{db} = 1$. Therefore, for every free variable y of Q_{ineq}, we neccesarily have that $\nu(y) = 1$ in a resulting valuation. As such, all valuations in $\texttt{Vals}(Q, db)$ are constant on the variables in $\overline{x}_2$ and duplicates can hence not occur when projecting on $\overline{x}_1$. The proposition then follows because we may enumerate $\texttt{Vals}(Q, db)$ with $\mathcal{O}(1)$ delay after linear time preprocessing by assumption. □

Proposition 10.11. *Let Q be a CQ over σ and $\mathcal{K}$ a semiring such that the subsemiring generated by $\mathbb{0}_\mathcal{K}$ and $\mathbb{1}_\mathcal{K}$ is non-trivial and zero-sum free. If $\texttt{Eval}(Q, \sigma, \mathcal{K}) \in \texttt{Enum}(\| db \|, 1)$ then $\texttt{Eval}(Q, \sigma, \mathbb{B}) \in \texttt{Enum}(\| db \|, 1)$.*

Proof. Let $\mathcal{K} = (K, \oplus, \odot, \mathbb{0}_\mathcal{K}, \mathbb{1}_\mathcal{K})$ be a a semiring such that the subsemiring generated by $\mathbb{0}_\mathcal{K}$ and $\mathbb{1}_\mathcal{K}$ is non-trivial and zero-sum free. We reduce $\texttt{Eval}(Q, \sigma, \mathbb{B})$ to $\texttt{Eval}(Q, \sigma, \mathcal{K})$ as follows. Let $db_\mathbb{B}$ be a $\mathbb{B}$-database over σ, input to $\texttt{Eval}(Q, \sigma, \mathbb{B})$. Construct the $\mathcal{K}$-database $db_\mathcal{K}$, input to $\texttt{Eval}(Q, \sigma, \mathcal{K})$ by setting

- for every relation symbol $R \in \sigma$

$$[\![R]\!]_{db_\mathcal{K}} : \overline{d} \mapsto \begin{cases} \mathbb{1}_\mathcal{K} \text{ if } R^{db_\mathbb{B}}(\overline{d}) = \texttt{t} \\ \mathbb{0}_\mathcal{K} \text{ otherwise.} \end{cases}$$

- $db_\mathcal{K}(c) := db_\mathbb{B}(c)$ for every constant symbol c.

Note that this takes time $\mathcal{O}(\| db_\mathbb{B} \|)$.

By assumption, $\texttt{Eval}(Q, \sigma, \mathcal{K}) \in \texttt{Enum}(\| db \|, 1)$ thus the set $\texttt{Vals}(Q, db_\mathcal{K})$ can be enumerated with constant delay after a linear time preprocessing. Now, it suffices to do the following: for every outputed tuple $(\nu, [\![Q]\!]_{db_\mathcal{K}}(\nu))$ we output $(\nu, \texttt{t})$. This forms a constant delay enumeration for the set $\texttt{Vals}(Q, db_\mathbb{B})$, since we claim that for every valuation ν we have $(\nu, \texttt{t}) \in \texttt{Vals}(Q, db_\mathbb{B})$ if and only if $(\nu, k) \in \texttt{Vals}(Q, db_\mathcal{K})$ for some $\mathbb{0}_\mathcal{K} \neq k \in K$. We next argue why this claim holds.

Let $\varphi = \exists \overline{y}. R_1(\overline{z}_1) \wedge \cdots \wedge R_n(\overline{z}_m) \wedge w_1 \leq c_1 \wedge \cdots \wedge \overline{w}_m \leq c_m$ be the body of $Q(\overline{x})$ and assume $\nu \colon \overline{x}$.

First, if $(\nu, [\![Q]\!]_{db_\mathcal{K}}(\nu)) \in \texttt{Vals}(Q, db_\mathcal{K})$ then in particular $[\![Q]\!]_{db_\mathcal{K}}(\nu) \neq \mathbb{0}_\mathcal{K}$. Because

$$\begin{aligned} [\![Q]\!]_{db_\mathcal{K}}(\nu) &= \bigoplus_{\mu \colon var(\varphi) \text{ s.t.} \mu|_{\overline{x}} = \nu} [\![\varphi]\!]_{db_\mathcal{K}}(\mu) \\ &= \bigoplus_{\mu \colon var(\varphi) \text{ s.t.} \mu|_{\overline{x}} = \nu} R_1^{db_\mathcal{K}}(\mu|_{\overline{z}_1}) \odot \cdots \odot R_n^{db_\mathcal{K}}(\mu|_{\overline{z}_n}) \odot \\ &\qquad [\![w_1 \leq c_1]\!]_{db_\mathcal{K}}(\mu|_{w_1}) \odot \cdots \odot [\![w_m \leq c_m]\!]_{db_\mathcal{K}}(\mu|_{w_m}) \end{aligned}$$

we know in particular that there exists some μ such that

$$R_1^{db_\mathcal{K}}(\mu|_{\overline{z}_1}) \odot \cdots \odot R_n^{db_\mathcal{K}}(\mu|_{\overline{z}_n}) \odot [\![w_1 \leq c_1]\!]_{db_\mathcal{K}}(\mu|_{w_1}) \odot \cdots \odot [\![w_m \leq c_m]\!]_{db_\mathcal{K}}(\mu|_{w_m}) \neq \mathbb{0}_\mathcal{K}.$$

This is because in any semiring, a sum of terms can be non-$\mathbb{0}$ only if one term is non-$\mathbb{0}$. Moreover, a product is non-$\mathbb{0}$ only if all of its factors are non-$\mathbb{0}$. By construction, $R_i^{db_\mathcal{K}}(\mu|_{\overline{z}_i}) \neq \mathbb{0}$ for all $i = 1, \ldots, n$ and $[\![w_j \leq c_j]\!]_{db_\mathcal{K}}(\mu|_w) \neq \mathbb{0}_\mathcal{K}$ for all $j = 1, \ldots, m$ only if $R_i^{db_\mathbb{B}}(\mu|_{\overline{z}_i}) = \mathtt{t}$ and $[\![w_j \leq c_j]\!]_{db_\mathbb{B}}(\mu|_w) = \mathtt{t}$ for all i and j. Thus $(\nu, \mathtt{t}) \in \mathtt{Vals}(Q, db_\mathbb{B})$.

Second, if $(\nu, \mathtt{t}) \in \mathtt{Vals}(Q, db_\mathbb{B})$ then in particular $[\![Q]\!]_{db_\mathbb{B}}(\nu) = \mathtt{t}$. Because

$$[\![Q]\!]_{db_\mathbb{B}}(\nu) = \bigvee_{\mu:\ var(\varphi)\ \text{s.t.}\mu|_{\overline{x}}=\nu} R_1^{db_\mathbb{B}}(\mu|_{\overline{z}_1}) \wedge \cdots \wedge R_n^{db_\mathbb{B}}(\mu|_{\overline{z}_n}) \wedge [\![w_1 \leq c_1]\!]_{db_\mathbb{B}}(\mu|_{w_1}) \wedge \cdots \wedge [\![w_m \leq c_m]\!]_{db_\mathbb{B}}(\mu|_{w_m})$$

this implies there is some valuation μ such that

$$R_1^{db_\mathbb{B}}(\mu|_{\overline{z}_1}) \wedge \cdots \wedge R_n^{db_\mathbb{B}}(\mu|_{\overline{z}_n}) \wedge [\![w_1 \leq c_1]\!]_{db_\mathbb{B}}(\mu|_{w_1}) \wedge \cdots \wedge [\![w_m \leq c_m]\!]_{db_\mathbb{B}}(\mu|_{w_m}) = \mathtt{t}.$$

In particular, $R_i^{db_\mathbb{B}}(\mu|_{\overline{z}_i}) = \mathtt{t}$ for all $i = 1, \ldots, n$ and $[\![w_j \leq c_j]\!]_{db_\mathbb{B}}(\mu|_{w_j}) = \mathtt{t}$ for $j = 1, \ldots, m$. Then, by construction of $db_\mathcal{K}$ we have $R_i^{db_\mathcal{K}}(\mu|_{\overline{z}_i}) = \mathbb{1}_\mathcal{K}$ for all $i = 1, \ldots, n$ and $[\![w_j \leq c_j]\!]_{db_\mathcal{K}}(\mu|_{w_j}) = \mathbb{1}_\mathcal{K}$ for $j = 1, \ldots, m$. Hence,

$$R_1^{db_\mathcal{K}}(\mu|_{\overline{z}_1}) \odot \cdots \odot R_n^{db_\mathcal{K}}(\mu|_{\overline{z}_n}) \odot [\![w_1 \leq c_1]\!]_{db_\mathcal{K}}(\mu|_{w_1}) \odot \cdots \odot [\![w_m \leq c_m]\!]_{db_\mathcal{K}}(\mu|_{w_m}) = \mathbb{1}_\mathcal{K}.$$

Thus, since the subsemiring generated by $\mathbb{0}_\mathcal{K}$ and $\mathbb{1}_\mathcal{K}$ is non-trivial and zero-sum free,

$$[\![Q]\!]_{db_\mathcal{K}}(\nu) = \bigoplus_{\mu:\ var(\varphi)\ \text{s.t.}\mu|_{\overline{x}}=\nu} [\![\varphi]\!]_{db_\mathcal{K}}(\mu) \neq \mathbb{0}_\mathcal{K}$$ □

Recall that by convention we only consider commutative and non-trivial semirings throughout the paper. As such when we consider an arbitrary semiring $\mathcal{K}$, then we assume that this semiring is commutative and non-trivial. For such semirings, it is straightforward to check that the sub-semiring generated by $\mathbb{0}$ and $\mathbb{1}$ is also commutative and non-trivial. (We note that we have only explicitly add the "non-trivial" requirement in the previous proposition because the proof uses this property.) Given this discussion, we can now prove Theorem 10.8.

Proof of Theorem 10.8. Let $\mathtt{split}(Q) = (Q_{\text{rel}}, Q_{\text{ineq}})$ and assume $\mathtt{Eval}(Q, \sigma, \mathcal{K}) \in \mathtt{Enum}(\| db \|, 1)$. Because the subsemiring of $\mathcal{K}$ generated by $\mathbb{0}_\mathcal{K}$ and $\mathbb{1}_\mathcal{K}$ is non-trivial and zero-sum free, $\mathtt{Eval}(Q, \sigma, \mathbb{B}) \in \mathtt{Enum}(\| db \|, 1)$ holds by Proposition 10.11. This implies $\mathtt{Eval}(Q_{\text{rel}}, \sigma, \mathbb{B}) \in \mathtt{Enum}(\| db \|, 1)$, due to Proposition 10.10. Theorem 9.1 yields that Q_{rel} is free-connex, unless either the Sparse Boolean Matrix Multiplication, the Triangle Detection, or the $(k, k+1)$-Hyperclique conjecture is false. Finally, since Q_{rel} is free-connex if and only if Q is free-connex (Proposition 10.3), this immediately implies that Q is free-connex, unless one of these hypothesis is false. □

Lower bound for fc-MATLANG queries. We now proceed to prove Corollary 10.9. A necessary ingredient to prove the lower bound is to have *enumeration-suited* matrix-to-relational encodings. Let $\mathbf{Q} := \mathbf{H} = e$ be a conj-MATLANG query. A matrix-to-relational encoding *Rel* is *enumeration-suited* for $\mathbf{Q}$ if

- *Rel*($\mathbf{A}$) is a nullary relation if $\mathbf{A}$ is of scalar type and a unary relation if $\mathbf{A}$ is of vector type (but not scalar), for every $\mathbf{A} \in \mathcal{S}(\mathbf{Q}) \setminus \mathbf{H}$.
- *Rel*($\mathbf{H}$) is a binary relation.

One can easily see that for every $\mathbf{Q}$ there exists some matrix-to-relational encoding *Rel* that is enumeration-suited for $\mathbf{Q}$ (i.e., choose the natural encoding that maps scalars to nullary relations, vectors to unary relations, and matrices to binary relations). Therefore, to prove Corollary 10.9 we may always choose a *Rel* that is enumeration-suited, and apply the following proposition to move from the matrix setting to the relational setting.

Proposition 10.12. *Let $\mathbf{Q} = \mathbf{H} := e$ be a* conj-MATLANG *query and let Rel be an enumeration-suited matrix-to-relational encoding for $\mathbf{Q}$. If $\mathbf{Q}$ is constant-disjoint then there exists a constant-disjoint CQ that simulates $\mathbf{Q}$ w.r.t. Rel.*

Proof. Assume that e, when converted into prenex normal form, is of the form $e = \Sigma \mathbf{x}, \mathbf{y}, \mathbf{v}_1, \ldots, \mathbf{v}_k.\ \mathbf{s}_1 \times \cdots \times \mathbf{s}_n \times \mathbf{x} \cdot \mathbf{y}^T$ and that this prenex normal form is constant-disjoint. Let $\overline{\mathbf{v}} = \mathbf{x}, \mathbf{y}, \mathbf{v}_1, \ldots, \mathbf{v}_k$.

The proof proceeds in two steps. (i) We first show that because *Rel* is enumeration-suited, the CQ Q *with equality atoms* simulating $\mathbf{Q}$ that is obtained through the translation given in the proof Proposition 4.2, is constant-disjoint. (ii) Then we show that eliminating equality atoms from Q yields a CQ Q' that is still constant disjoint.

Formally for point (i) to make sense, we say that a CQ with equality atoms is constant-disjoint if satisfies the same conditions as for normal CQs, but with the modification that now an inequality atom $z \leq c$ is covered if variable z is covered: it appears in a relational atom or it appears in an equality atom $z = w$ and w is covered. Note that this definition is recursive. For example, in the formula $\exists u, v, w.A(x, u) \wedge u = v \wedge v = w$ we have that u, v, w are covered.

We first prove (i). The Q CQ simulating $\mathbf{Q}$ obtained by Proposition 4.2 is of the form $Q\colon H(x, y) \leftarrow \exists v_1, \ldots, v_k.a_1 \wedge \cdots \wedge a_n \wedge x \leq \alpha \wedge y \leq \beta$ where each a_i is determined as follows, for $1 \leq i \leq n$. Let non-bold symbol A denote *Rel*($\mathbf{A}$), and let non-bold w be the FO variable selected respectively for $\mathbf{w} \in \{\mathbf{x}, \mathbf{y}, \mathbf{v}_1, \ldots, \mathbf{v}_k\}$.

- If $\mathbf{s}_i = \mathbf{v}^T \cdot \mathbf{A} \cdot \mathbf{w}$ then:
  - $a_i := A(v, w)$ if A is binary. Note that since *Rel* is enumeration suited, this case can only happen if $\mathbf{A}\colon (\alpha, \beta)$ with $\alpha \neq 1$ and $\beta \neq 1$. Due to well-typedness, $\mathbf{v}\colon (\alpha, 1)$ and $\mathbf{w}\colon (\beta, 1)$.
  - $a_i := A(v) \wedge w \leq 1$ if A is unary and $\mathbf{A}\colon (\alpha, 1)$. Note that since *Rel* is enumeration suited, necessarily $\alpha \neq 1$. For later use we remark that since $\mathbf{s}_i$ is well-typed, $\mathbf{v}\colon (\alpha, 1)$ and $\mathbf{w}\colon (1, 1)$ in this case.
  - $a_i := v \leq 1 \wedge A(w)$ if A is unary and $\mathbf{A}\colon (1, \alpha)$. Since *Rel* is enumeration-suited, necessarily $\alpha \neq 1$. For later use we remark that since $\mathbf{s}_i$ is well-typed, $\mathbf{v}\colon (1, 1)$ and $\mathbf{w}\colon (\alpha, 1)$.
  - $a_i := v \leq 1 \wedge w \leq 1 \wedge A()$ if A is nullary and $\mathbf{A}\colon (1, 1)$. Note that necessarily $\mathbf{v}\colon (1, 1)$ and $\mathbf{w}\colon (1, 1)$ in this case.
- If $\mathbf{s}_i = \mathbf{v}^T \cdot \mathbf{w}$ with $\mathbf{v}\colon (\gamma, 1)$ and $\mathbf{w}\colon (\gamma, 1)$ then:
  - $a_i := v \leq \gamma$ if $\mathbf{v} = \mathbf{w}$.

– $a_i \coloneqq v \leq \gamma \wedge w \leq \gamma \wedge v = w$ if $\mathbf{v} \neq \mathbf{w}$.

Recall that a CQ Q is constant-disjoint if (1) for all covered inequalities $z \leq c$ we have $c \neq 1$, and (2) for all pairs $(v \leq c, y \leq d)$ in Q of covered inequality $w \leq c$ and non-covered inequality $w \leq d$ if $c = d$ then $w \notin \mathit{free}(Q)$.

We start by showing that in Q condition (1) always holds. Consider an inequality $z_1 \leq 1$ in Q. This necessarily appears in some subformula a_{i_1} with $\mathbf{s}_{i_1} = \mathbf{v}_{i_1}^T \cdot \mathbf{A} \cdot \mathbf{w}_{i_1}$ and $\mathbf{z}_1 \in \{\mathbf{v}_{i_1}, \mathbf{w}_{i_1}\}$. By inspection of the construction above we observe that necessarily $\mathbf{z}_1 \colon (1,1)$. We further observe that z_1 does not appear in a relational atom in the subformula a_{i_1}. Hence, if z is covered in Q, it must be because there are other subformulas $a_{i_2}, \ldots, a_{i_\ell}$ that contain equality atoms $z_{j-1} = z_j$ for $2 \leq j \leq \ell$, as well as a subformula $a_{i_{\ell+1}}$ where z_j occurs in a relational atom. The corresponding subexpressions $\mathbf{s}_{i_j}$ of e with $1 \leq j \leq \ell$ are necessarily of the form $\mathbf{v}_{i_j}^T \cdot \mathbf{w}_{i_j}$ with $\mathbf{z}_{i_j} \in \{\mathbf{v}_{i_j}, \mathbf{w}_{i_j}\}$. Moreover, $\mathbf{s}_\ell$ must be of the form $\mathbf{v}_{\ell+1}^T \cdot \mathbf{B} \cdot \mathbf{w}_{\ell+1}$ with $\mathbf{z}_{i_\ell} \in \{\mathbf{v}_{i_\ell}, \mathbf{w}_{i_\ell}\}$. Because each $\mathbf{s}_{i_j}$ is well-typed and all $\mathbf{v}_{i_j}, \mathbf{w}_{i_j}$ are vector variables, it follows that each $\mathbf{z}_{i_j}$ has type $(1,1)$ for $1 \leq j \leq \ell$. This yields a contradiction: it is readily verified that when the construction generates a relational atom containing a variable z_ℓ, that variable cannot have type $(1,1)$.

We prove that if $\mathbf{Q}$ is constant-disjoint, then property (2) holds in Q. We also do this by contradiction. Suppose it does not hold, i.e., suppose there is a pair $(v \leq c, w \leq c)$ in Q of covered inequality $v \leq c$ and non-covered inequality $w \leq c$ and assume that w is free in Q. Then $w \in \{x, y\}$ and since $v \leq c$ is covered it is the case that there is some $a_i \in \{A(v,z), A(z,v), A(v) \wedge z \leq 1, z \leq 1 \wedge A(v)\}$. Because Rel is enumeration-suited, we have that $\mathbf{s}_i \in \{\mathbf{v}^T \cdot \mathbf{A} \cdot \mathbf{u}, \mathbf{u}^T \cdot \mathbf{A} \cdot \mathbf{v}\}$ with $\mathbf{v} \colon (\gamma, 1)$ with $\gamma \neq 1$. Hence we have a pair $(\mathbf{v}, \mathbf{w})$ where $\mathbf{v} \colon (\gamma, 1)$ is covered, $\mathbf{w} \colon (\gamma, 1)$ is not covered and $\mathbf{w} \in \{\mathbf{x}, \mathbf{y}\}$, which means that $\mathbf{Q}$ is not constant-disjoint, a contradiction.

For step (ii), let Q' be the equivalent CQ after unification and removal of equality atoms on Q. First note that since Q is well-typed w.r.t. Rel^{-1} also Q' is well-typed w.r.t Rel^{-1} since we only unify variables, and since all unified variables must have had the same type in Q by definition of the FO^+ type system. Further observe that because our definition of constant-disjointness in Q takes equality into account, and because Q' is obtained by unifying variables that must be equal, also Q' must necessarily be constant-disjoint. □

We are ready to prove Corollary 10.9.

Proof of Corollary 10.9. Let Rel be a matrix-to-relational encoding scheme over $\mathcal{S}$ that is enumeration-suited for $\mathbf{Q}$. Let $\mathit{Mat} = \mathit{Rel}^{-1}$ be its inverse relational-to-matrix encoding.

We apply Proposition 10.12 on $\mathbf{Q}$ to obtain binary CQ Q over $\sigma = \mathit{Rel}(\mathcal{S})$ that simulates $\mathbf{Q}$ w.r.t. Rel. We know that Q is binary, self-join free and constant-disjoint. We next modify Q into a query Q' over σ such that the following properties hold:

(P1) Q' continues to simulate $\mathbf{Q}$ w.r.t. Rel;
(P2) Q' continues to be constant-disjoint;
(P3) $\mathtt{Eval}(Q', \sigma, \mathcal{K}) \in \mathtt{Enum}(\| db \|, 1)$.

This suffices to prove the proposition. Indeed, by Theorem 10.8 Q' is necessarily free-connex unless unless either the Sparse Boolean Matrix Multiplication, the Triangle Detection, or the $(k, k+1)$-Hyperclique conjecture is false. If Q' is free-connex, then by Corollary 6.2 there exists a fc-MATLANG query $\mathbf{Q}'$ that simulates Q' w.r.t $\mathit{Mat} = \mathit{Rel}^{-1}$. Consequently, $\mathbf{Q}$ and $\mathbf{Q}'$ are then equivalent.

Intuitively, the modification of Q into Q' is necessary to obtain property (P3): the fact that $\texttt{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K}) \in \texttt{Enum}(\|\mathcal{I}\|, 1)$ only implies that we may evaluate Q with linear time preprocessing and constant delay *on databases that encode some matrix instance*. To apply Theorem 10.8, by contrast we need to show that we can evaluate Q on *arbitrary databases*. The crux will be that we obtain Q' by adding extra inequalities to Q, such that evaluating Q' on an arbitrary database yields the same result as evaluating it (and Q, $\mathbf{Q}$) on some matrix instance.

The definition of Q' is as follows. Assume Q is of the form $Q : H(\overline{y}) \leftarrow \exists \overline{z}.\psi$ with ψ quantifier free. For every relational atom $\alpha = R(\overline{x})$ of ψ we define a formula φ_α as follows. This formula is a conjunction of inequalities:

- If $\alpha = R(x_1, x_2)$ with R binary and $\mathit{Mat}(R)\colon (\beta, \gamma)$ then $\varphi_\alpha := x_1 \leq \mathit{Rel}(\beta) \wedge x_2 \leq \mathit{Rel}(\gamma)$. Note that, by definition of Rel, $\mathit{Rel}(\alpha) \neq 1 \neq \mathit{Rel}(\beta)$.
- If $\alpha = R(x_1)$ with R unary then by definition of Rel either $\mathit{Mat}(R)\colon (\beta, 1)$ or $\mathit{Mat}(R)\colon (1, \beta)$ for some size symbol $\beta \neq 1$. Then take $\varphi_\alpha := x_1 \leq \mathit{Rel}(\beta)$. Note that by definition of matrix-to-relational encoding schemes, $\mathit{Rel}(\beta) \neq 1$ since $\beta \neq 1$.
- If $\alpha = R()$ with R nullary then by definition of Rel we have $\mathit{Mat}(R)\colon (1, 1)$ and we take φ_α the empty formula (which is equivalent to true).

Then we define Q' to be the CQ

$$Q'\colon H(\overline{y}) \leftarrow \exists z.(\psi \wedge \bigwedge_{\alpha \text{ rel. atom in } \psi} \varphi_\alpha).$$

Note that Q' continues to simulate $\mathbf{Q}$ over Rel (property P1). Indeed, for any matrix instance $\mathcal{I}$ over $\mathcal{S}$ we know that $\texttt{Answer}(\mathbf{Q}, \mathcal{I}) = \texttt{Answer}(Q, \mathit{Rel}(\mathcal{I}))$. By definition, $\mathit{Rel}(\mathcal{I})$ is consistent with $\mathit{Rel}^{-1} = \mathit{Mat}$, and therefore the tuples in $\mathit{Rel}(\mathcal{I})$ vacuously satisfy all the extra inequalities that we have added to Q to obtain Q'. As such, $\texttt{Answer}(Q, \mathit{Rel}(\mathcal{I})) = \texttt{Answer}(Q', \mathit{Rel}(\mathcal{I}))$. Thus, Q' continues to simulate $\mathbf{Q}$.

Also note that Q' is constant-disjoint (property P2). Indeed:

- We have only added covered inequalities to Q to obtain Q'. Any such covered inequality that we have added was of the form $x \leq c$ with $c \neq 1$. Therefore, since Q does not have covered inequalities of the form $x \leq 1$, neither does Q'.
- We did not add any non-covered inequalities to Q to obtain Q'. Moreover, Q' has the same free variables as Q. Therefore, all non-covered inequalities $y \leq c$ in Q' continue to be such that y is not free in Q', unless $c = 1$. Hence for all pairs $(x \leq c, y \leq d)$ of covered inequality $x \leq c$ and non-covered inequality $y \leq d$ with $d \neq 1$ in Q' we have that y is not free in Q' as required for constant-disjointness.

Finally, we show that $\texttt{Eval}(Q', \sigma, \mathcal{K}) \in \texttt{Enum}(\|\, db \,\|, 1)$ (property P3), as claimed. We do this by reducing $\texttt{Eval}(Q', \sigma, \mathcal{K})$ to $\texttt{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ as follows. Let db be a $\mathcal{K}$-database over σ, input to $\texttt{Eval}(Q', \sigma, \mathcal{K})$. First, create the database db' that is equal to db except that for every relation symbol $R \in \sigma$ that is not mentioned in Q we set $R^{db'}$ to empty. This can clearly be done in linear time. Note that, if R is not mentioned in Q, then the answer of Q on db is independent of the contents of R^{db}. Therefore, $\texttt{Answer}(Q', db) = \texttt{Answer}(Q', db')$. Subsequently, compute db'', the atomic reduction of db' w.r.t. Q. Computing db'' can be done in time $\mathcal{O}(\|\, db' \,\|) = \mathcal{O}(\|\, db \,\|)$. By Claim 10.4 we have $\texttt{Answer}(Q', db) = \texttt{Answer}(Q', db') = \texttt{Answer}(Q', db'')$. Now verify the following claim: db'' is consistent w.r.t. $\mathit{Mat} = \mathit{Rel}^{-1}$. This is because we have added, for each matrix symbol $\mathbf{A}$ with $\mathit{Rel}(\mathbf{A})$ occurring in Q, the inequalities that are required by consistency as atomic inequalities to Q'. (The matrix

symbols $\mathbf{A}$ with $Rel(\mathbf{A})$ not occurring in Q are empty in db' and hence vacuously consistent w.r.t. Mat.) Since db'' is consistent with Mat we have that $Mat(db'')$ is a matrix instance over $\mathcal{S}$ and hence a valid input to $\mathbf{Q}$. Then, because Q' simulates $\mathbf{Q}$, we have $\texttt{Answer}(Q', db) = \texttt{Answer}(Q', db'') = \texttt{Answer}(\mathbf{Q}, Mat(db''))$. The last equality holds because the head of Q' is a binary relation since Rel is enumeration-suited for $\mathbf{Q}$.

Because $\texttt{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ is in $\texttt{Enum}(\|\mathcal{I}\|, 1)$ it hence follows that with additional preprocessing of time $\mathcal{O}(\|\, Mat(db'')\,\|) = \mathcal{O}(\|\, db\,\|)$ we may enumerate $\texttt{Answer}(Q', db'') = \texttt{Answer}(Q', db)$ with constant delay, as desired. □

11. Efficient evaluation of q-hierarchical queries

This last section derives our algorithmic results for qh-CQ and qh-MATLANG. qh-CQ is the subclass of CQ that allows efficient evaluation in a dynamic setting, where insertion and deletion of tuples are admitted. Then, similar to the previous section, we aim to lift these algorithmic results to qh-MATLANG.

We start by introducing the dynamic setting for query evaluation, and then study the upper and lower bounds.

11.1. 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.

As the reader may have noticed, updates allow to modify the contents of relations, but not of constant symbols. This is because an update to a constant translates as an update to the dimension value of a matrix within the equivalent linear algebra setting. And updates in the linear algebra setting affect only entry values, not dimensions. An interesting line of future work is to consider dimension updates.

Dynamic enumeration problems. A *dynamic enumeration problem* is an enumeration problem P together with a set U of *updates* on P's inputs: a set of operations such that applying an update $u \in U$ to an input I of P yields a new input to P, denoted $I + u$. An algorithm A *solves* a dynamic enumeration problem $D = (P, U)$ if it solves the enumeration problem P (computing, for each input I a data structure in a preprocessing phase from which $P(I)$ may be enumerated) and, moreover, it is possible, for every input I and update u to update the data structure that A computed on I during the update phase to a data structure for $I+u$. By definition, the latter updated data structure hence allows to enumerate $P(I + u)$. The preprocessing time and enumeration delay of A are defined as for normal

(static) enumeration problems. The *update time* of A is an upper bound on the time needed to update the data structure for I into one for $I + u$.

In this document, we will consider two versions of the dynamic query evaluation problem for conjunctive query Q over σ under semiring $\mathcal{K}$:

- The *unbounded* version, $\texttt{DynEval}(Q, \sigma, \mathcal{K})$ which is the dynamic enumeration problem $(\texttt{Eval}(Q, \sigma, \mathcal{K}), U)$ with U the set of all single-tuple insertions and deletions over $\mathcal{K}$ and σ.
- The *bounded* version, denoted $\texttt{DynEval}(Q, \sigma, \mathcal{K}, M)$, where $M \in \mathbb{N}_{\geq 0}$ is a constant positive natural number, and which is defined as follows.

  Define $\texttt{Eval}(Q, \sigma, \mathcal{K}, M)$ to be the bounded version of the *static* query evaluation problem: this is the collection of all pairs (I, O) with $I = db$ with db a $\mathcal{K}$-database over σ *such that any data value d occurring in db is at most M, i.e., $d \leq M$*[7], and $O = \texttt{Answer}(Q, db)$. Hence, $\texttt{Eval}(Q, \sigma, \mathcal{K}, M)$ is the evaluation problem of Q on $\mathcal{K}$-databases whose values in the active domain are bounded by M.

  Then the bounded dynamic evaluation problem $\texttt{DynEval}(Q, \sigma, \mathcal{K}, M)$ is the dynamic enumeration problem $(\texttt{Eval}(Q, \sigma, \mathcal{K}, M), U)$ with U the set of all single-tuple updates $\texttt{insert}(R, \overline{d}, k)$ or $\texttt{delete}(R, \overline{d})$ where every value $d \in \overline{d}$ satisfies $d \leq M$.

Clearly, one can check that $\texttt{DynEval}(Q, \sigma, \mathcal{K}) = \bigcup_{M=1}^{\infty} \texttt{DynEval}(Q, \sigma, \mathcal{K}, M)$.

In an analogous manner, in the matrix setting for a query $\mathbf{Q}$ over $\mathcal{S}$ we define the dynamic query evaluation problem $\texttt{DynEval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ and $\texttt{DynEval}(\mathbf{Q}, \mathcal{S}, \mathcal{K}, M)$ of $\texttt{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ and $\texttt{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K}, M)$, respectively. The allowed updates in these problems are of the form $\texttt{insert}(\mathbf{A}, i, j, k)$, which sets $\mathbf{A}_{i,j} := \mathbf{A}_{i,j} \oplus k$, and $\texttt{delete}(\mathbf{A}, i, j)$ which sets $\mathbf{A}_{i,j} := \mathbb{0}$.

Dynamic complexity classes. For functions f, g, and h from the natural numbers to the positive reals we define $\texttt{DynEnum}(f, g, h)$ to be the class of dynamic enumeration problems (P, U) for which there exists an enumeration algorithm A such that for every input I it holds that the preprocessing time is $\mathcal{O}(f(\| I \|))$, the delay is $\mathcal{O}(g(\| I \|))$, where $\| I \|$ denotes the size of I. Moreover, processing update u on current input I takes time $\mathcal{O}((h(\| I \|, \| u \|))$. Note that, since the inputs in a dynamic evaluation problem $\texttt{DynEval}(Q, \sigma, \mathcal{K})$ or $\texttt{DynEval}(Q, \sigma, \mathcal{K}, M)$ consists only of the database and not the query, we hence measure complexity in *data complexity* as is standard in the literature [BDG07, IUV+20, BGS20, BKS17]. In what follows we set the size $\| u \|$ of a single-tuple update u to 1: each update is of constant size, given the fixed vocabulary σ.

Note that, since $\texttt{DynEval}(Q, \sigma, \mathcal{K}) = \bigcup_{M=1}^{\infty} \texttt{DynEval}(Q, \sigma, \mathcal{K}, M)$ if $\texttt{DynEval}(Q, \sigma, \mathcal{K}) \in \texttt{DynEnum}(\| db \|, 1, 1)$ then also $\texttt{DynEval}(Q, \sigma, \mathcal{K}, M) \in \texttt{DynEnum}(\| db \|, 1, 1)$, for every M. Conversely, if $\texttt{DynEval}(Q, \sigma, \mathcal{K}, M) \notin \texttt{DynEnum}(\| db \|, 1, 1)$ for some M, then also $\texttt{DynEval}(Q, \sigma, \mathcal{K}) \notin \texttt{DynEnum}(\| db \|, 1, 1)$. Specifically, we will use the bounded evaluation problem $\texttt{DynEval}(Q, \sigma, \mathcal{K}, M)$ to obtain our lower bounds in the relational setting. Analogously, we focus on the bounded evaluation problem $\texttt{DynEval}(\mathbf{Q}, \mathcal{S}, \mathcal{K}, M)$ to obtain our lower bounds in the matrix setting.

Complexity hypothesis and known results. Like the authors of [HKNS15, BKS17] we use the following hypothesis concerning the hardness of dynamic problems as hypothesis for obtaining conditional lower bounds:

- The *Online Boolean Matrix-Vector Multiplication (OMv) problem*: given an $n \times n$ boolean matrix A, compute $A \cdot v_1, \ldots, A \cdot v_n$ sequentially for $n \times 1$ vectors $v_1, \ldots, v_n$. It is required

[7] Recall that the set of data values that may appear in database tuples equals $\mathbb{N}_{\geq 0}$.

that the result of $A \cdot v_{i-1}$ must already be computed in order to access v_i. The *OMv conjecture* states that this problem cannot be solved in time $\mathcal{O}(n^{3-\varepsilon})$ for any $\varepsilon > 0$.

Theorem 11.1 [BKS17]**.** *Let Q be a CQ$^{\not\leq}$ over σ.*

- *If Q is q-hierarchical, then $\texttt{DynEval}(Q, \sigma, \mathbb{B}) \in \texttt{DynEnum}(\| db \|, 1, 1)$.*
- *If Q is a query without self joins and $\texttt{DynEval}(Q, \sigma, \mathbb{B}, M) \in \texttt{DynEnum}(\| db \|, 1, 1)$ for every $M \in \mathbb{N}_{>0}$, then Q is q-hierarchical unless the OMv conjecture is false.*

11.2. **Upper bounds for q-hierarchical queries.** Similar as 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 $\mathbb{N}$.

The main result of this subsection is the following.

Theorem 11.2. *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 $\mathcal{S}$.*

We now prove Theorem 11.2 in two steps. We first transfer the upper bound of CQ$^{\not\leq}$ queries over $\mathbb{B}$-databases to CQ$^{\not\leq}$ queries over $\mathcal{K}$-databases. We then generalize the latter to CQ queries over $\mathcal{K}$-databases.

Proposition 11.3. *Let Q be a q-hierarchical CQ$^{\not\leq}$ over σ and $\mathcal{K}$ a sum-maintainable semi-integral domain. Then $\texttt{DynEval}(Q, \sigma, \mathcal{K}) \in \texttt{DynEnum}(\| db \|, 1, 1)$.*

Proof. Note that since Q is q-hierarchical it has a guarded query plan (T, N) by Proposition 7.2. This implies in particular that Q is also free-connex. By Proposition 10.2 we hence know that $\texttt{Eval}(Q, \sigma, \mathcal{K}) \in \texttt{Enum}(\| db \|, 1)$. As such, there is an algorithm A that, given db builds a data structure in linear time from which $\texttt{Answer}(Q, db)$ can be enumerated with constant delay. We will show that for any update u to db this data structure can also be updated in $\mathcal{O}(1)$ into a data structure for $\texttt{Answer}(Q, db + u)$. Because, in the proof of Proposition 10.2 the algorithm (and data structure) depends on the shape of Q and (T, N) we make a corresponding case analysis here.

(1) If Q is a full-join, i.e., no variable is quantified, we reduce to the Boolean case as follows. Assume that $Q = H(\overline{z}) \leftarrow R_1(\overline{z}_1) \wedge \cdots \wedge R_n(\overline{z}_n)$. In $\mathcal{O}(\| db \|)$ time construct $\texttt{Bool}(db)$, the $\mathbb{B}$-database obtained from db defined as:

$$R^{\texttt{Bool}(db)} \colon \overline{d} \mapsto \begin{cases} \texttt{t} \text{ if } R^{db}(\overline{d}) \neq \mathbb{0} \\ \texttt{f} \text{ otherwise,} \end{cases}$$

for every $R \in \sigma$. Furthermore, preprocess db by creating lookup tables such that for every relation R and tuple $\overline{a}$ of the correct arity we can retrieve the annotation $R^{db}(\overline{a})$ in time $\mathcal{O}(|\overline{a}|)$ time. Note that since σ is fixed, $|\overline{a}|$ is constant. This retrieval is hence $\mathcal{O}(1)$ in data complexity. It is well-known that such lookup tables can be created in $\mathcal{O}(\| db \|)$ time in the RAM model.

Finally, invoke the algorithm for $\texttt{DynEval}(Q, \sigma, \mathbb{B})$ on $\texttt{Bool}(db)$. By Theorem 11.1, this algorithm has $\mathcal{O}(\| \texttt{Bool}(db) \|) = \mathcal{O}(\| db \|)$ preprocessing time, after which we may enumerate $\texttt{Answer}(Q, \texttt{Bool}(db))$ with $\mathcal{O}(1)$ delay and maintain this property under updates to $\texttt{Bool}(db)$ in $\mathcal{O}(1)$ time. We have shown in Proposition 10.2 that using the enumeration procedure for $\texttt{Answer}(Q, \texttt{Bool}(db))$ and the lookup tables, we may also enumerate $\texttt{Answer}(Q, db)$ with constant delay. We do not repeat this argument here.

Given a single-tuple update u we update the lookup tables and the data structure maintained by $\texttt{DynEval}(Q, \sigma, \mathbb{B})$ as follows.

- If $u = \texttt{insert}(R, \overline{d}, k)$ then lookup $\overline{d}$ in the lookup table for R and retrieve its old annotation ℓ (if the lookup table does not contain ℓ, set $\ell = \mathbb{0}$). There are two cases to consider:
  - If $k + \ell = \mathbb{0}$ then the tuple $\overline{d}$ is now deleted from R^{db+u} and we similarly remove it from $\texttt{Bool}(db + u)$ by issuing the update $u' = \texttt{delete}(R, \overline{d})$ to $\texttt{Bool}(db)$ (and $\texttt{DynEval}(Q, \sigma, \mathbb{B})$).
  - If $k + \ell \neq \mathbb{0}$ then the tuple is certainly present in R^{db+u} inserted and similarly make sure it is in $\texttt{Bool}(db + u)$ by issuing $u' = \texttt{insert}(R, \overline{d}, \texttt{t})$ to $\texttt{Bool}(db)$ (and $\texttt{DynEval}(Q, \sigma, \mathbb{B})$). Note that if the tuple was already in $\texttt{Bool}(db)$ then u' has no-effect since we are working in the boolean semiring there.
- If $u = \texttt{delete}(R, \overline{d})$ then delete $\overline{d}$ from the lookup table and issue the update $u' = \texttt{delete}(R, \overline{d})$ to $\texttt{Bool}(db)$ and $\texttt{DynEval}(Q, \sigma, \mathbb{B})$.

Note that in all cases, after issuing u' we obtain $\texttt{Bool}(db+u)$ as desired, so that after this update using the enumeration procedure for $\texttt{Answer}(Q, \texttt{Bool}(db + u))$ and the lookup tables, we will enumerate $\texttt{Answer}(Q, db + u)$ with constant delay.

(2) If the guarded query plan (T, N) is such that N consists of a single node $N = \{r\}$, then by Lemma 7.3 we may assume without loss of generality that for every node n with two children c_1, c_2 in T we have $var(n) = var(c_1) = var(c_2)$. Intuitively, the algorithm will work as follows. We show that for any database db we may compute $\texttt{Answer}(Q, db)$ in $\mathcal{O}(\| db \|)$ time. Because we can simply store this result, we may certainly enumerate it with constant delay. Then, by also storing the subresults computed during the computation of $\texttt{Answer}(Q, db)$ we show that we can also maintain $\texttt{Answer}(Q, db)$ and all subresults in $\mathcal{O}(1)$ time under updates.

Formally the data structure used by $\texttt{DynEval}(Q, \sigma, \mathcal{K})$ will be the query result itself, plus some extra lookup tables. In particular, given input database db we:

- compute, for every node $n \in T$ the $\texttt{Answer}(\varphi[T, n], db)$. We have shown in Proposition 10.2 that we may compute $\texttt{Answer}(\varphi[T, n], db)$ in linear time, for every node n. We do not repeat the argument here. Since Q (and hence T) is fixed there are a constant number of nodes in T. So this step takes linear time overall.
- convene that we store $\texttt{Answer}(\varphi[T, n], db)$ for every node n by creating a lookup table such that for any tuple $(\overline{a}, k) \in \texttt{Answer}(\varphi[T, n], db)$ we can retrieve k in $\mathcal{O}(1)$ time given $\overline{a}$. It is well-known that the lookup tables can be made such that may we enumerate the entries $(\overline{a}, k)$ of this lookup table with constant delay. It is well-known that such

lookup tables can be created in time linear in $\| \texttt{Answer}(\varphi[T,n], db) \| = \mathcal{O}(\| db \|)$. So also this step takes linear time.

Note that by definition $Q \equiv \varphi[T,r]$; therefore the root represents $\texttt{Answer}(Q, db)$ as desired, and can be used to enumerate the result with constant delay.

It remains to show that given an update u to db we can update $\texttt{Answer}(\varphi[T,n]db,)$ into $\texttt{Answer}(\varphi[T,n]db + u,)$ in $\mathcal{O}(1)$ time, for every node n. In order to be able to show this, we store one auxiliary lookup table L_n for all nodes n that have a single child c. This auxiliary table is defined as follows. By Lemma 5.8 if n has a single child c then $\varphi[T,n] \equiv \exists \overline{y}.\varphi[T,c]$ with $\overline{y} = var(c) \setminus var(n)$. The lookup table L_n stores, for each $(\overline{d}, k)$ in $\texttt{Answer}(\varphi[T,n], db)$ a pointer to a data structure $\mathcal{D}_{\overline{d}}$ that represents the multiset

$$\{\{k \mid (\overline{a}, k) \in \texttt{Answer}(\varphi[T_c, c], db), \overline{a}|_{var(r)} = \overline{d}\}\}.$$

Because $\mathcal{K}$ is sum-maintainable, and because $\texttt{Answer}(\varphi[T_c, c], db)$ is linear in db, this extra lookup table can be computed in linear time.[8]

We now verify that for every single-tuple update u to db and every node n we can update $\texttt{Answer}(\varphi[T,n]db,)$ into $\texttt{Answer}(\varphi[T,n]db + u,)$ in $\mathcal{O}(1)$ time. The proof is by induction on the height of n in T. (I.e., the length of longest path from n to a leaf.)

- When n is a leaf, it is an atom, say $R(\overline{x})$. Upon update u, if u is of the form $\texttt{insert}(R, \overline{a}, k)$ or $\texttt{delete}(R, \overline{a})$ then we update the lookup table that represents $\texttt{Answer}(\varphi[T,n]db,)$ in the obvious way. Such updates take $\mathcal{O}(1)$ in the RAM model. If u pertains to a relation other then R, we simply ignore it.
- When n is an interior node with a single child c then by Lemma 5.8 we have $\varphi[T,n] \equiv \exists \overline{y}.\varphi[T,c]$ with $\overline{y} = var(c) \setminus var(r)$. When we receive update u to db, by inductive hypothesis this yields a bounded number of single-tuple updates u_c to $A = \texttt{Answer}(\varphi[T,c], db)$ which we may compute in $\mathcal{O}(1)$ time. We translate these updates u_c into updates u' to apply to $\texttt{Answer}(\varphi[T,n], db)$:
  - If $u_c = \texttt{insert}(\overline{a}, k)$. Then apply u_c to A and update the auxiliary lookup table L_n by applying $\texttt{ins}(k, \mathcal{D}_{\overline{d}})$ with $\overline{d} = \overline{a}|_{var(r)}$. Furthermore, update $\texttt{Answer}(\varphi[T,n], db)$ by applying $u' = \texttt{insert}(\overline{d}, k)$ to the lookup table representing it.
  - If $u_c = \texttt{delete}(\overline{a})$. Before applying u_c to $A = \texttt{Answer}(\varphi[T,c], db)$, look up the old annotation k of $\overline{a}$ in A. Then apply u_c to A and update L_n by applying $\texttt{del}(k, \mathcal{D}_{\overline{d}})$ with $\overline{d} = \overline{a}|_{var(r)}$. If $\texttt{query}(\mathcal{D}_{\overline{d}}) = \mathbb{0}$, then u effectively causes $\overline{d}$ to be removed from $\texttt{Answer}(\varphi[T,n], db)$. Hence, the update u' to apply to (the lookup table representing) $\texttt{Answer}(\varphi[T,n], db)$ is $u' = \texttt{delete}(\overline{d})$. Otherwise, $\texttt{query}(\mathcal{D}_{\overline{d}}) \neq \mathbb{0}$ and the update u' to apply is $u' = \texttt{insert}(\overline{d}, k)$.
- Node n has two children c_1 and c_2. By Lemma 5.9 we have $\varphi[T,n] \equiv \varphi[T,c_1] \wedge \varphi[T,c_2]$. Note that this denotes an intersection since $var(n) = var(c_1) = var(c_2)$. When we receive update u to db, by inductive hypothesis this yields a bounded number of single-tuple updates to $A_1 = \texttt{Answer}(\varphi[T,c_1], db)$ and $A_2 = \texttt{Answer}(\varphi[T,c_1], db)$ which we may compute in $\mathcal{O}(1)$ time. We translate these updates to A_1 and A_2 into updates to $\texttt{Answer}(\varphi[T,n]db,)$ as follows. We only show the reasoning for updates to A_1. The

[8] Initially, the lookup table is empty. Loop over the entries $(\overline{a}, k)$ of $\texttt{Answer}(\varphi[T_c, c], db)$ one by one. For each such tuple, compute $\overline{d} = \overline{a}|_{var(r)}$ and lookup $\overline{d}$ in L. If it is present with data structure $\mathcal{D}_{\overline{d}}$ then apply $\texttt{ins}(k, \mathcal{D}_{\overline{d}})$, hence updating $\mathcal{D}_{\overline{d}}$. If it is not present, initialize $\mathcal{D}_{\overline{d}}$ to empty; add k to it, and insert $(\overline{d}, \mathcal{D}_{\overline{d}})$ to the lookup table.

reasoning for updates to A_2 is similar. Let u_1 be update to A_1. We translate this into update u' to apply to $\texttt{Answer}(\varphi[T,n], db)$ as follows:
- if $u_1 = \texttt{insert}(\overline{d}, k_1)$ then use the lookup table on A_2 to find the annotation ℓ of $\overline{d}$ in A_2. (If $\overline{d}$ does not appear in A_2 then take $\ell = \mathbb{0}$). If $k_1 \odot \ell \neq \mathbb{0}$ then apply $u' = \texttt{insert}(d, k_1 \odot \ell)$ to $\texttt{Answer}(\varphi[T,n], db)$; otherwise there is no update u' to apply.
- if $u_1 = \texttt{delete}(\overline{d})$ then apply $u' = \texttt{delete}(d)$ to $\texttt{Answer}(\varphi[T,n], db)$.

(3) When Q is not full and N has more than one node we reduce to the previous two cases because $\varphi[T,N] \equiv \varphi[T_1,n_1] \wedge \cdots \wedge \varphi[T_l,n_l]$ where $F = \{n_1,\ldots,n_l\}$ is the frontier of N, i.e, the nodes without children in N, and $T_1,\ldots,T_l$ are the subtrees of T rooted at $n_1,\ldots,n_l$ respectively. Hence, we may enumerate $\texttt{Answer}(\varphi[T,N], db)$ with constant delay by first fully computing the full join $\varphi[T_1,n_1] \wedge \cdots \wedge \varphi[T_l,n_l]$ on these subresults. By item (2) above, each $A_i := \texttt{Answer}(\varphi[T_i,n_i], db)$ can be computed in linear time from db. Furthermore, each $\texttt{Answer}(\varphi[T_i,n_i], db)$ can be maintained under updates in $\mathcal{O}(1)$ time by item (2) above. By item (1) the result of the final full join can be enumerated with constant delay after linear time processing on A_i. In addition, we can maintain this property under updates in $\mathcal{O}(1)$ time by item (1) above, since this is a full join and $\varphi[T,N]$ is q-hierarchical. Therefore, the entire result $\texttt{Answer}(\varphi[T,N], db)$ can be enumerated with constant delay after linear time processing, and be maintained under updates in $\mathcal{O}(1)$ time. □

Next, we show how to extend Proposition 11.3 when queries have inequalities. Towards this goal, note that the conditions for a CQ to be q-hierarchical are imposed over relational atoms. Hence, the following result is straightforward (recall the definition of $\texttt{split}(Q)$ given in Section 10).

Corollary 11.4. *Let Q be a CQ and $\texttt{split}(Q) = (Q_{rel}, Q_{ineq})$. Then Q is q-hierarchical if and only if Q_{rel} is q-hierarchical.*

Then, we can exploit the split of Q between Q_{rel} and Q_{ineq} to evaluate efficiently any q-hierarchical query Q as follows.

Proposition 11.5. *Let Q be a CQ over σ and $\texttt{split}(Q) = (Q_{rel}, Q_{ineq})$ and $\mathcal{K}$ a semi-integral domain. If $\texttt{DynEval}(Q_{rel}, \sigma, \mathcal{K}) \in \texttt{DynEnum}(\| db \|, 1, 1)$ then $\texttt{DynEval}(Q, \sigma, \mathcal{K}) \in \texttt{DynEnum}(\| db \|, 1, 1)$.*

Proof. Let σ be the vocabulary of Q and db be an arbitrary $\mathcal{K}$-database over σ. Since $\texttt{DynEval}(Q_{\text{rel}}, \sigma, \mathcal{K}) \in \texttt{DynEnum}(\| db \|, 1, 1)$ we have $\texttt{Eval}(Q_{\text{rel}}, \sigma, \mathcal{K}) \in \texttt{Enum}(\| db \|, 1)$ and consequently $\texttt{Eval}(Q, \sigma, \mathcal{K}) \in \texttt{Enum}(\| db \|, 1)$, by Proposition 10.7.

Note that Proposition 10.7 shows that $\texttt{Eval}(Q, \sigma, \mathcal{K}) \in \texttt{Enum}(\| db \|, 1)$ by first making input database db atomically consistent with Q and then invoking $\texttt{Eval}(Q_{\text{rel}}, \sigma, \mathcal{K})$ on the atomically reduced database db'. It is then possible to enumerate $\texttt{Vals}(Q, db') = \texttt{Vals}(Q, db)$ with constant delay, by enumerating $\texttt{Vals}(Q_{\text{rel}}, db') = \texttt{Vals}(Q_{\text{rel}}, db)$ with constant delay. In other words, the data structure of $\texttt{Eval}(Q, \sigma, \mathcal{K})$ is simply that of $\texttt{Eval}(Q_{\text{rel}}, \sigma, \mathcal{K})$.

Hence, to obtain the proposition, it suffices to translate a given update u to db into an update u' to db' such that $db' + u'$ is the atomic reduction of $db + u$. This allows us to update the data structure $\texttt{Eval}(Q_{\text{rel}}, \sigma, \mathcal{K})$ in $\mathcal{O}(1)$ time by assumption, and still be able to enumerate $\texttt{Vals}(Q_{\text{rel}}, db' + u') = \texttt{Vals}(Q_{\text{rel}}, db + u)$. We translate the update as follows

- If $u = \texttt{insert}(R, \overline{d}, k)$ then for each atom $R(\overline{x})$ in Q with $\overline{d} \models \overline{x}$ check that the inequality constraints imposed by $Q[R(\overline{x})]$ are satisfied. If this is the case, apply $u' = u$ to db'. Otherwise, no update needs to be done to db'.
- If $u = \texttt{delete}(R, \overline{d})$ then apply $u' = u$ to db'. □

We finally have all the machinery to prove Theorem 11.2.

Proof of Theorem 11.2. Let $\texttt{split}(Q) = (Q_{\text{rel}}, Q_{\text{ineq}})$. Since Q is q-hierarchical, we have that Q_{rel} is q-hierarchical by Corollary 11.4. Then, by Proposition 11.3, $\texttt{DynEval}(Q_{\text{rel}}, \sigma, \mathcal{K}) \in \texttt{DynEnum}(\| db \|, 1, 1)$ since $\mathcal{K}$ is a sum-maintainable semi-integral domain. Hence we get that $\texttt{DynEval}(Q, \sigma, \mathcal{K}) \in \texttt{DynEnum}(\| db \|, 1, 1)$ because of Proposition 11.5.

We proceed to prove the second part of the result, i.e., achieve the same processing complexity for the matrix evaluation problem $\texttt{DynEval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$. Let $\mathbf{Q}$ be a qh-MATLANG query over $\mathcal{S}$ and $\mathcal{I}$ and instance over $\mathcal{S}$. Fix a matrix-to-relational schema encoding *Rel* over $\mathcal{S}$ such that the head atom of Q is binary. Since in particular $\mathbf{Q}$ is also a fc-MATLANG query, by Theorem 10.1 $[\![\mathbf{Q}]\!](\mathcal{I})$ can be enumerated with constant delay after a preprocessing that runs in time $\mathcal{O}(\| \mathcal{I} \|)$. Now, use Theorem 8.2 to compute the q-hierarchical relational query Q over σ that is equivalent to $\mathbf{Q}$ under *Rel*. This is in constant time in data complexity. Next, let u be a update u to $\mathcal{I}$. We define a series of single tuple updates $u_1, \ldots, u_n$ to $\mathit{Rel}(\mathcal{I})$ such that $\mathit{Rel}(\mathcal{I}) + u_1 + \ldots + u_n = \mathit{Rel}(\mathcal{I}) + u = \mathit{Rel}(\mathcal{I}_u)$. This can be done in $\mathcal{O}(\| u \|)$ time. Because Q is q-hierarchical, $\texttt{DynEval}(Q, \sigma, \mathcal{K})$ can be evaluated dynamically with constant-time update and constant-delay, and hence we can enumerate each $\texttt{Answer}(Q, \mathit{Rel}(\mathcal{I}))$, $\texttt{Answer}(Q, \mathit{Rel}(\mathcal{I}) + u_1)$ up to $\texttt{Answer}(Q, \mathit{Rel}(\mathcal{I}) + u_1 + \ldots + u_n)$ with constant delay. This holds because processing each update u_i one at a time takes time $\mathcal{O}(1)$. After $\mathcal{O}(\| u \|)$ time, the former yields constant delay enumeration for $\texttt{Answer}(\mathbf{Q}, \mathcal{I}_u)$. □

11.3. **Lower bounds for q-hierarchical queries.** Similar as for free-connex CQ, we can extend the lower bound in [BKS17] 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 [HKNS15]. For this, recall the definition of constant-disjoint introduced in Section 10.

Theorem 11.6. *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.*

To prove Theorem 11.6 it suffices to show that the statement holds in the bounded problem $\texttt{DynEval}(Q, \sigma, \mathbb{B}, M)$, for every $M \in \mathbb{N}_{>0}$. Towards that goal, we first reduce the $\texttt{DynEval}(Q, \sigma, \mathbb{B}, M)$ problem into the $\texttt{DynEval}(Q_{\text{rel}}, \sigma, \mathbb{B}, M)$ problem assuming that Q is constant-disjoint. Subsequently, we reduce $\texttt{DynEval}(Q, \sigma, \mathcal{K}, M)$ to $\texttt{DynEval}(Q, \sigma, \mathbb{B}, M)$.

Proposition 11.7. *Let $M \in \mathbb{N}_{>0}$ and let Q be a CQ over σ with $\texttt{split}(Q) = (Q_{rel}, Q_{ineq})$. If Q is constant-disjoint and $\texttt{DynEval}(Q, \sigma, \mathbb{B}, M) \in \texttt{DynEnum}(\| db \|, 1, 1)$ then $\texttt{DynEval}(Q_{rel}, \sigma, \mathbb{B}, M) \in \texttt{DynEnum}(\| db \|, 1, 1)$.*

Proof. The proof uses the same ideas as the proof of Proposition 10.10, but specialized to the dynamic setting, and utilizing that M is a bound on the active domain of the databases that we will evaluate Q_{rel} on.

Specifically, we reduce $\texttt{DynEval}(Q_{\text{rel}}, \sigma_{\text{rel}}, \mathbb{B}, M)$ to $\texttt{DynEval}(Q, \sigma, \mathbb{B}, M)$. Denote the free variables of Q, Q_{rel} and Q_{ineq} by $\overline{x}$, $\overline{x}_1$ and $\overline{x}_2$, respectively. Note that $\overline{x}_1$ and $\overline{x}_2$ are

disjoint and $var(\overline{x}) = var(\overline{x}_1) \cup var(\overline{x}_2)$. Consider an arbitrary database db_{rel} over σ, input to $\texttt{DynEval}(Q_{\text{rel}}, \sigma, \mathbb{B}, M)$. In particular, all values in the active domain of db are bounded by M. Construct the database db, input to $\texttt{DynEval}(Q, \sigma, \mathbb{B}, M)$ as follows.

- $R^{db} = R^{db_{\text{rel}}}$ for each relation symbol $R \in \sigma$;
- $db(c) := M$ for all constant symbols $c \in \sigma$ that occur in a covered inequality in Q. Note that $c \neq 1$ because Q is constant-disjoint;
- $db(c) := 1$ for all other constant symbols $c \in \sigma$.

This is well-defined because Q is constant-disjoint. The computation of db is clearly in $\mathcal{O}(\| db_{\text{rel}} \|)$ and $\| db \| = \| db_{\text{rel}} \|$. We call db the *extension* of db_{rel}.

Because M is an upper bound on the domain values occurring in db_{rel} we have shown in Proposition 10.10 that $\texttt{Answer}(Q_{\text{rel}}, db_{\text{rel}}) = \texttt{Answer}(Q_{\text{rel}}, db)$ and, that moreover, we may enumerate $\texttt{Answer}(Q_{\text{rel}}, db)$ with constant delay using the constant-delay enumeration for $\texttt{Answer}(Q, db)$, which exists because $\texttt{DynEval}(Q, \sigma, \mathbb{B}, M) \in \texttt{DynEnum}(\| db \|, 1, 1)$.

Hence, to obtain the proposition, it suffices to show that we can translate any given update u to db_{rel} into an update u' to db such that $db + u'$ is the extension of $db_{\text{rel}} + u$. Because $\texttt{DynEval}(Q, \sigma, \mathbb{B}, M) \in \texttt{DynEnum}(\| db \|, 1, 1)$ this allows us to update the data structure of $\texttt{Eval}(Q_{\text{rel}}, \sigma, \mathbb{B}, M)$ in $\mathcal{O}(1)$, and still be able to enumerate $\texttt{Answer}(Q, db + u')$, which yields the enumeration of $\texttt{Answer}(Q, db_{\text{rel}} + u)$ with constant delay. We reason as follows. Let u be a single-tuple update to db_{rel} such that all data values occurring in u are bounded by M. Then simply apply u to db. Clearly, $db + u$ is the extension of $db_{\text{rel}} + u$. □

Proposition 11.8. *Let Q be a CQ over σ and $\mathcal{K}$ a semiring such that the subsemiring generated by $\mathbb{0}_{\mathcal{K}}$ and $\mathbb{1}_{\mathcal{K}}$ is non-trivial and zero-sum free. Let $M \in \mathbb{N}_{>0}$. If $\texttt{DynEval}(Q, \sigma, \mathcal{K}, M) \in \texttt{DynEnum}(\| db \|, 1, 1)$ then $\texttt{DynEval}(Q, \sigma, \mathbb{B}, M) \in \texttt{DynEnum}(\| db \|, 1, 1)$.*

Proof. Let $\mathcal{K} = (K, \oplus, \odot, \mathbb{0}_{\mathcal{K}}, \mathbb{1}_{\mathcal{K}})$ be a a semiring such that the subsemiring generated by $\mathbb{0}_{\mathcal{K}}$ and $\mathbb{1}_{\mathcal{K}}$ is non-trivial and zero-sum free. The proof uses the same ideas as the proof Proposition 10.11, but specialized to the dynamic setting, with bounded evaluation.

Specifically, we reduce $\texttt{DynEval}(Q, \sigma, \mathbb{B}, M)$ to $\texttt{DynEval}(Q, \sigma, \mathcal{K}, M)$ as follows. Let $db_{\mathbb{B}}$ be a $\mathbb{B}$-database over σ, input to $\texttt{DynEval}(Q, \sigma, \mathbb{B}, M)$. Construct the $\mathcal{K}$-database $db_{\mathcal{K}}$, input to $\texttt{DynEval}(Q, \sigma, \mathcal{K}, M)$ by setting

- for every relation symbol $R \in \sigma$

$$[\![R]\!]_{db_{\mathcal{K}}} : \overline{d} \mapsto \begin{cases} \mathbb{1}_{\mathcal{K}} \text{ if } R^{db_{\mathbb{B}}}(\overline{d}) = \mathtt{t} \\ \mathbb{0}_{\mathcal{K}} \text{ otherwise.} \end{cases}$$

- $db_{\mathcal{K}}(c) := db_{\mathbb{B}}(c)$ for every constant symbol c.

We call $db_{\mathcal{K}}$ the $\mathcal{K}$ version of $db_{\mathbb{B}}$ Note that this takes time $\mathcal{O}(\| db_{\mathbb{B}} \|)$. Then run $\texttt{DynEval}(Q, \sigma, \mathcal{K}, M)$ on $db_{\mathcal{K}}$, which takes linear time. We have shown in the proof of Proposition 10.11 that we may then enumerate $\texttt{Answer}(Q, db_{\mathbb{B}})$ with constant delay using the constant-delay enumeration procedure for $\texttt{Answer}(Q, db_{\mathcal{K}})$ provided by $\texttt{DynEval}(Q, \sigma, \mathcal{K}, M)$. By assumption, $\texttt{Eval}(Q, \sigma, \mathcal{K}) \in \texttt{Enum}(\| db \|, 1)$ thus the set $\texttt{Vals}(Q, db_{\mathcal{K}})$ can be enumerated with constant delay after a linear time preprocessing.

Hence, to obtain the proposition, it suffices to show that we can translate any given update u to $db_{\mathbb{B}}$ into an update u' to $db_{\mathcal{K}}$ such that $db_{\mathcal{K}} + u'$ yields the $\mathcal{K}$-version of $db_{\mathbb{B}} + u$. Because $\texttt{DynEval}(Q, \sigma, \mathcal{K}, M) \in \texttt{DynEnum}(\| db \|, 1, 1)$ this allows us to update the data structure of $\texttt{DynEval}(Q, \sigma, \mathbb{B}, M)$ in $\mathcal{O}(1)$, and still be able to enumerate $\texttt{Answer}(Q, db_{\mathcal{K}} + u')$, which yields the enumeration of $\texttt{Answer}(Q, db_{\text{rel}} + u)$ with constant delay.

We reason as follows. Let u be a single-tuple update to $db_{\mathbb{B}}$ such that all data values occurring in u are bounded by M.

- If $u = \texttt{insert}(R, \overline{d}, \texttt{t})$ and $\overline{d}$ does not occur in $R^{db_{\mathcal{K}}}$ (meaning that $\overline{d}$ did not occur in $db_{\mathbb{B}}$ before), then set $u' = \texttt{insert}(R, \overline{d}, \mathbb{1}_{\mathcal{K}})$.
- If $u = \texttt{insert}(R, \overline{d}, \texttt{f})$ then $db + u$ is the same as db, and we hence issue no update u' to $db_{\mathcal{K}}$.
- If $u = \texttt{delete}(R, \overline{d})$ then the update $u' = \texttt{delete}(R, \overline{d})$. □

As argued, the next theorem clearly implies Theorem 11.6.

Theorem 11.9. *Let Q be a CQ over σ without self-joins and constant-disjoint and $\mathcal{K}$ a semiring such that the subsemiring generated by $\mathbb{0}_{\mathcal{K}}$ and $\mathbb{1}_{\mathcal{K}}$ is non-trivial and zero-sum free. $\texttt{DynEval}(Q, \sigma, \mathcal{K}, M) \in \texttt{DynEnum}(\| db \|, 1, 1)$ for every $M \in \mathbb{N}_{>0}$, then Q is q-hierarchical, unless the OMv conjecture is false.*

Proof. Let $\texttt{split}(Q) = (Q_{\text{rel}}, Q_{\text{ineq}})$. Since $\texttt{DynEval}(Q, \sigma, \mathcal{K}, M) \in \texttt{DynEnum}(\| db \|, 1, 1)$ for every M, by Proposition 11.8 we have that $\texttt{DynEval}(Q, \sigma, \mathbb{B}, M) \in \texttt{DynEnum}(\| db \|, 1, 1)$ for every M, because the subsemiring generated by $\mathbb{0}_{\mathcal{K}}$ and $\mathbb{1}_{\mathcal{K}}$ is non-trivial zero-sum free. Proposition 11.7 states then that $\texttt{DynEval}(Q_{\text{rel}}, \sigma, \mathbb{B}, M) \in \texttt{DynEnum}(\| db \|, 1, 1)$ for every M, since Q is constant disjoint. Because $Q_{\text{rel}} \in \text{CQ}^{\not\leq}$, by Theorem 11.1, Q_{rel} is q-hierarchical unless the OMv conjecture is false. Given that Q_{rel} is q-hierarchical if and only if Q is q-hierarchical, the former directly implies that Q is q-hierarchical unless the OMv conjecture is false. □

11.4. **Lower bounds for** qh-MATLANG **queries.** Similarly to the lower bound in the free-connex case, we show that Theorem 11.9 transfers to conj-MATLANG.

Corollary 11.10. *Let $\mathbf{Q}$ be a* conj-MATLANG *query over $\mathcal{S}$ such that $\mathbf{Q}$ does not repeat matrix symbols and $\mathbf{Q}$ is 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 $\texttt{DynEval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ 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.*

Similarly as in the relational case, we rely on the fact that the following theorem clearly implies Corollary 11.10: indeed, if $\texttt{DynEval}(\mathbf{Q}, \mathcal{S}, \mathcal{K}) \in \texttt{Enum}(\| \mathcal{I} \|, 1)$ then trivially $\texttt{DynEval}(\mathbf{Q}, \mathcal{S}, \mathcal{K}, M) \in \texttt{Enum}(\| \mathcal{I} \|, 1)$ for every M, in which case the following proposition yields the claim of Corollary 11.10.

Theorem 11.11. *Let $\mathbf{Q}$ be a* conj-MATLANG *query over $\mathcal{S}$ such that $\mathbf{Q}$ does not repeat matrix symbols and $\mathbf{Q}$ is constant-disjoint. Let $\mathcal{K}$ be a semiring such that the subsemiring generated by $\mathbb{0}_{\mathcal{K}}$ and $\mathbb{1}_{\mathcal{K}}$ is non-trivial and zero-sum free. If $\texttt{DynEval}(\mathbf{Q}, \mathcal{S}, \mathcal{K}, M) \in \texttt{Enum}(\| \mathcal{I} \|, 1)$ for every $M \in \mathbb{N}_{>0}$, then $\mathbf{Q}$ is equivalent to a* qh-MATLANG *query, unless the OMv conjecture is false.*

Proof. Let *Rel* be a matrix-to-relational encoding scheme over $\mathcal{S}$ such that it is enumeration-suited for $\mathbf{Q}$ (see Section 10.2). Let $Mat = Rel^{-1}$ be its inverse relational-to-matrix encoding.

We apply Proposition 10.12 on $\mathbf{Q}$ to obtain binary CQ Q over $\sigma = Rel(\mathcal{S})$ that simulates $\mathbf{Q}$ w.r.t. *Rel*. We know that Q is binary, self-join free and constant-disjoint. We next modify Q into a query Q' over σ such that the following properties hold:

(P1) Q' continues to simulate $\mathbf{Q}$ w.r.t. *Rel*;

(P2) Q' continues to be constant-disjoint;
(P3) $\texttt{DynEval}(Q', \sigma, \mathcal{K}, M) \in \texttt{DynEnum}(\| db \|, 1, 1)$, for every $M \in \mathbb{N}_{>0}$.

This suffices to prove the proposition. Indeed, by Theorem 11.9 Q' is necessarily q-hierarchical unless the OMv conjecture fails. If Q' is q-hierarhical, then by Corollary 6.2 there exists a fc-MATLANG query $\mathbf{Q}'$ that simulates Q' w.r.t $Mat = Rel^{-1}$. Consequently, $\mathbf{Q}$ and $\mathbf{Q}'$ are then equivalent.

Intuitively, the modification of Q into Q' is necessary to obtain property (P3): the fact that $\texttt{DynEval}(\mathbf{Q}, \mathcal{S}, \mathcal{K}, M) \in \texttt{Enum}(\| \mathcal{I} \|, 1)$ only implies that we may dynamically evaluate Q in $\texttt{DynEnum}(\| db \|, 1, 1)$ *on databases that encode some matrix instance*. To apply Theorem 11.9, by contrast we need to show that we can dynamically evaluate Q on *arbitrary databases*. The crux will be that we obtain Q' by adding extra inequalities to Q, such that evaluating Q' on an arbitrary database yields the same result as evaluating it (and Q, $\mathbf{Q}$) on some matrix instance.

The definition of Q' is as follows. Assume Q is of the form $Q : H(\overline{y}) \leftarrow \exists \overline{z}.\psi$ with ψ quantifier free. For every relational atom $\alpha = R(\overline{x})$ of ψ we define a formula φ_α as follows. This formula is a conjunction of inequalities:

- If $\alpha = R(x_1, x_2)$ with R binary and $Mat(R)\colon (\beta, \gamma)$ then $\varphi_\alpha := x_1 \leq Rel(\beta) \wedge x_2 \leq Rel(\gamma)$. Note that, by definition of Rel, $Rel(\alpha) \neq 1 \neq Rel(\beta)$.
- If $\alpha = R(x_1)$ with R unary then by definition of Rel either $Mat(R)\colon (\beta, 1)$ or $Mat(R)\colon (1, \beta)$ for some size symbol $\beta \neq 1$. Then take $\varphi_\alpha := x_1 \leq Rel(\beta)$. Note that by definition of matrix-to-relational encoding schemes, $Rel(\beta) \neq 1$ since $\beta \neq 1$.
- If $\alpha = R()$ with R nullary then by definition of Rel we have $Mat(R)\colon (1, 1)$ and we take φ_α the empty formula (which is equivalent to $\texttt{t}$).

Then we define Q' to be the CQ

$$Q'\colon H(\overline{y}) \leftarrow \exists z.(\psi \wedge \bigwedge_{\alpha \text{ rel. atom in } \psi} \varphi_\alpha).$$

Note that Q' continues to simulate $\mathbf{Q}$ over Rel (property P1). Indeed, for any matrix instance $\mathcal{I}$ over $\mathcal{S}$ we know that $\texttt{Answer}(\mathbf{Q}, \mathcal{I}) = \texttt{Answer}(Q, Rel(\mathcal{I}))$. By definition, $Rel(\mathcal{I})$ is consistent with $Rel^{-1} = Mat$, and therefore the tuples in $Rel(\mathcal{I})$ vacuously satisfy all the extra inequalities that we have dded to Q to obtain Q'. As such, $\texttt{Answer}(Q, Rel(\mathcal{I})) = \texttt{Answer}(Q', Rel(\mathcal{I}))$. Thus, Q' continues to simulate $\mathbf{Q}$.

Also note that Q' is constant-disjoint (property P2). Indeed:

- We have only added covered inequalities to Q to obtain Q'. Any such covered inequality that we have added was of the form $x \leq c$ with $c \neq 1$. Therefore, since Q does not have covered inequalities of the form $x \leq 1$, neither does Q'.
- We did not add any uncovered inequalities to Q to obtain Q'. Moreover, Q' has the same free variables as Q. Therefore, all uncovered inequalities $y \leq c$ in Q' continue to be such that y is not free in Q', unless $c = 1$. Hence for all pairs $(x \leq c, y \leq d)$ of covered inequality $x \leq c$ and non-covered inequality $y \leq d$ with $d \neq 1$ in Q' we have that y is not free in Q' as required for constant-disjointness.

Finally, we show that $\texttt{DynEval}(Q', \sigma, \mathcal{K}, M) \in \texttt{DynEnum}(\| db \|, 1, 1)$ for every $M \in \mathbb{N}_{>0}$ (property P3), as claimed. Fix $M \in \mathbb{N}_{>0}$ arbitrarily. We reduce $\texttt{DynEval}(Q', \sigma, \mathcal{K}, M)$ to $\texttt{DynEval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ (which is an *unbounded* evaluation problem) as follows. Let db be a $\mathcal{K}$-database over σ, input to $\texttt{DynEval}(Q', \sigma, \mathcal{K}, M)$. First, create the database db' that is equal to db except that for every relation symbol $R \in \sigma$ that is not mentioned in Q we set

$R^{db'}$ to empty. This can clearly be done in linear time. Note that, if R is not mentioned in Q, then it is also not mentioned in Q', and hence of Q' on db is independent of the contents of R^{db}. Therefore, $\texttt{Answer}(Q', db) = \texttt{Answer}(Q', db')$. Subsequently, compute db'', the atomic reduction of db' w.r.t. Q. Computing db'' can be done in time $\mathcal{O}(\| db' \|) = \mathcal{O}(\| db \|)$. By Claim 10.4 we have $\texttt{Answer}(Q', db) = \texttt{Answer}(Q', db') = \texttt{Answer}(Q', db'')$. Now verify the following claim: db'' is consistent w.r.t. $Mat = Rel^{-1}$. This is because we have added, for each matrix symbol $\mathbf{A}$ with $Rel(\mathbf{A})$ occurring in Q, the inequalities that are required by consistency as atomic inequalities to Q'. (The matrix symbols $\mathbf{A}$ with $Rel(\mathbf{A})$ not occurring in Q are empty in db' and hence vacuously consistent w.r.t. Mat.) Since db'' is consistent with Mat we have that $Mat(db'')$ is a matrix instance over $\mathcal{S}$ and hence a valid input to $\mathbf{Q}$. Then, because Q' simulates $\mathbf{Q}$, we have $\texttt{Answer}(Q', db) = \texttt{Answer}(Q', db'') = \texttt{Answer}(\mathbf{Q}, Mat(db''))$. Because $\texttt{DynEval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ is in $\texttt{Enum}(\| \mathcal{I} \|, 1)$ it hence follows that with additional preprocessing of time $\mathcal{O}(\| Mat(db'') \|) = \mathcal{O}(\| db \|)$ we may enumerate $\texttt{Answer}(Q', db'') = \texttt{Answer}(Q', db)$ with constant delay, as desired.

In essence, therefore, the data structure needed to enumerate $\texttt{Answer}(Q', db)$ with constant delay is the data structure computed by $\texttt{DynEval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ when run on $Mat(db'')$. We next show how to maintain this data structure under updates u to db.

Specifically, we next show how we may translate in $\mathcal{O}(1)$ time an update u to db into an update u'' to db'' and an update u_M to $Mat(db'')$ so that (1) $db'' + u''$ yields the same database as when we would do the above procedure starting from $db + u$ instead of db: $(db + u)'' = db'' + u''$ and (2) applying u_M to $Mat(db'')$ yields $Mat(db'' + u'')$. Therefore, applying u_M to $\texttt{DynEval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$ will give us the same data structure as when it was run on $Mat(db'' + u'')$. From this we can again enumerate $\texttt{Answer}(Q', db'' + u'') = \texttt{Answer}(Q', db + u)$ with constant delay. This is because the output relation is binary since Rel is enumeration-suited for $\mathbf{Q}$. We reason as follows.

- If $u = \texttt{insert}(R, \overline{d}, k)$ with R not occurring in Q' then we ignore u, there are no updates u'' and u_M to be done.
- If $u = \texttt{insert}(R, \overline{d}, k)$ with R occurring in Q' then we check whether $\overline{d}$ is atomically consistent with $Q'[R(\overline{x})]$ for every atom $R(\overline{x})$ in Q'. This can be done in constant time, because Q' (and hence the number of inequalities to check) is fixed. If it is atomically consistent then $u' = u$ and u_M inserts k in $Mat(R)$ at the coordinates specified by $\overline{d}$.
- If $u = \texttt{delete}(R, \overline{d}, k)$ with R not occurring in Q' then we ignore u, there are no updates u'' and u_M to be done.
- If $u = \texttt{delete}(R, \overline{d}, k)$ with R occurring in Q' then we check whether $\overline{d}$ is atomically consistent with $Q'[R(\overline{x})]$ for every atom $R(\overline{x})$ in Q'. This can be done in constant time, because Q' is fixed. If it is atomically consistent then $u' = u$ and u_M sets the entry in $Mat(R)$ at the coordinates specified by $\overline{d}$ to $\mathbb{0}$. □

12. 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 [CK21]. 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 [BGS20, CS23].

References

[ABC+16] 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.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: http://webdam.inria.fr/Alice/.

[ASS+17] 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.

[BDG07] 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.

[BGdB20] 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.

[BGdBW19] 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.

[BGS20] 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.

[BHPS20] 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.

[BKS17] 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.

[Bra13] 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.

[CK21] 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.

[CS23] 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.

[DG07] 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.

[DSS14] 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.

[EBH+17] 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.

[ECK24] Idan Eldar, Nofar Carmeli, and Benny Kimelfeld. Direct access for answers to conjunctive queries with aggregation. In Graham Cormode and Michael Shekelyan, editors, *27th International Conference on Database Theory, ICDT 2024, March 25-28, 2024, Paestum, Italy*, volume 290 of *LIPIcs*, pages 4:1–4:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. URL: https://doi.org/10.4230/LIPIcs.ICDT.2024.4, doi:10.4230/LIPICS.ICDT.2024.4.

[ELB+17] 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.

[Fag83] Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3):514–550, 1983. doi:10.1145/2402.322390.

[Gee19] 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.

[GKT07] 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.

[GMRV21] 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.

[Gol13] Jonathan S Golan. *Semirings and their Applications*. Springer Science & Business Media, 2013.

[GR22] 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.

[HBY13] 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.

[HKNS15] 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.

[HW96] Udo Hebisch and Hans Joachim Weinert. Semirings and semifields. *Handbook of Algebra*, 1:425–462, 1996.

[HW98] Udo Hebisch and Hanns Joachim Weinert. *Semirings: algebraic theory and applications in computer science*, volume 5. World Scientific, 1998.

[IUV17] 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.

[IUV+20] 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.

[JLY+20] 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.

[KNOZ20] 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.

[KS13] 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.

[KVG+19] 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.

[LGG+18] 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.

[SEM+20] 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.

[SSV18] 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.

[WHS+20] 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.

[Yan81] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*, pages 82–94. IEEE Computer Society, 1981.

[YHZ+17] 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.

Appendix A. Omitted proofs from Section 2

Proof of Proposition 2.3.

Proof. Let φ be a safe FO^+ formula. First observe the following: every subformula ψ of φ that is itself not an equality atom, is also safe. This is because safety is defined "universally" on φ. Fix a database *db*. We prove that $[\![\varphi]\!]_{db}$ has finite support by induction on φ.

- Both $R(\overline{x})$ and $x \leq c$ have finite support by definition.
- $x = y$ by itself is not safe; this case hence cannot occur.
- If $\varphi = \varphi_1 \wedge \varphi_2$, we discern the following cases.
  - Neither φ_1 nor φ_2 are an equality atom $x = y$. Then both φ_1 and φ_2 are safe, and the result follows by induction hypothesis because an annotation computed by φ can only be non-zero if is obtained by multiplying non-zero annotations originating from φ_1 and φ_2, which both have finite support.
  - One of φ_1 or φ_2 is of the form $x = y$. Assume for the sake of presentation that $\varphi_2 = (x = y)$. Then, by definition of safety, at least one of x, y is in $\mathsf{rr}(\varphi_1)$. In particular, from the definition of $\mathsf{rr}(\cdot)$ it follows that φ_1 itself cannot be an equality atom. Therefore, φ_1 is safe and the induction hypothesis applies to it. We discern two further cases.
    - Both $x, y \in \mathsf{rr}(\varphi_1)$. Since $\mathsf{rr}(\varphi_1) \subseteq \mathit{free}(\varphi_1)$, it follows that $\mathit{free}(\varphi_1) = \mathit{free}(\varphi)$. The result follows directly from the induction hypothesis, since it follows that the support of $[\![\varphi]\!]_{db}$ is a subset of the support of $[\![\varphi_1]\!]_{db}$, which is finite by induction hypothesis.
    - Only one of $x, y \in \mathsf{rr}(\varphi_1)$. Assume w.l.o.g. that $x \in \mathsf{rr}(\varphi_1)$ but $y \notin \mathsf{rr}(\varphi_1)$. By induction hypothesis, $[\![\varphi_1]\!]_{db}$ has finite support. Now consider a valuation $\nu\colon \mathit{free}(\varphi)$ such that $[\![\varphi]\!]_{db}(\nu) \neq \mathbb{0}$ and define $\nu_1 = \nu|_{\mathit{free}(\varphi_1)}$. By definition of the semantics of $\wedge$ we have also $[\![\varphi_1]\!]_{db}(\nu_1) \neq \mathbb{0}$. Now note that ν_1 completely specifies ν: the only variable that is potentially in the domain of ν but not in the domain of ν_1 is y, but φ requires $x = y$. So, knowing $\nu(x)$, which is given by $\nu_1(x)$, we also know $\nu(y)$. We conclude that if there were an infinite number of distinct $\nu\colon \mathit{free}(\varphi)$ such that $[\![\varphi]\!]_{db}(\nu) \neq \mathbb{0}$ then there are also an infinite number of $\nu_1\colon \mathit{free}(\varphi_1)$ such that $[\![\varphi]\!]_{db}(\nu) \neq \mathbb{0}$, which cannot happen by induction hypothesis.
- If $\varphi = \varphi_1 \vee \varphi_2$, then neither φ_1 nor φ_2 can themselves be equality atoms. (If they were it would violate condition (1) of safety.). The result then straightforwardly follows from the fact that $\mathit{free}(\varphi_1) = \mathit{free}(\varphi_2) = \mathit{free}(\varphi)$ and from the induction hypothesis.
- If $\varphi = \exists y.\varphi'$ then φ' itself cannot be an equality atom: if it were it would violate condition (1) of safety. The result then straightforwardly follows from the induction hypothesis. □

Appendix B. Omitted proofs from Section 4

Proof of Lemma 4.1.

Proof. We prove the lemma by induction on e.

- If $e = \mathbf{A}\colon (\alpha, \beta)$ then fix $\mathbf{x}\colon (\alpha, 1)$ and $\mathbf{y}\colon (\beta, 1)$. Clearly, $e \equiv \Sigma \mathbf{x}, \mathbf{y}.\ \left(\mathbf{x}^T \cdot \mathbf{A} \cdot \mathbf{y}\right) \times \mathbf{x} \cdot \mathbf{y}^T$.
- If $e = \mathbf{w}\colon (\alpha, 1)$ with $\mathbf{w}$ a vector variable, then fix $\mathbf{x}\colon (\alpha, 1)$ and $\mathbf{y}\colon (1, 1)$. Clearly, $e \equiv \Sigma \mathbf{x}, \mathbf{y}.\ \left(\mathbf{x}^T \cdot \mathbf{w}\right) \times \mathbf{x} \cdot \mathbf{y}^T$.
- If $e = (e_1)^T\colon (\beta, \alpha)$ with $e_1\colon (\alpha, \beta)$. By inductive hypothesis we can assume that $e_1 \equiv \Sigma \mathbf{x}, \mathbf{y}, \mathbf{v}_1, \ldots, \mathbf{v}_k.\, \mathbf{s} \times \mathbf{x} \cdot \mathbf{y}^T$. Hence, $e \equiv \Sigma \mathbf{y}, \mathbf{x}, \mathbf{v}_1, \ldots, \mathbf{v}_k.\, \mathbf{s} \times \mathbf{y} \cdot \mathbf{x}^T$.
- If $e = e_1 \odot e_2$ with $e_1\colon (\alpha, \beta)$ and $e_2\colon (\alpha, \beta)$, then:
$$e_1 \odot e_2 = \Sigma \mathbf{x}, \mathbf{y}.\ \left(\left(\mathbf{x}^T \cdot e_1 \cdot \mathbf{y}\right) \times \left(\mathbf{x}^T \cdot e_2 \cdot \mathbf{y}\right)\right) \times \left(\mathbf{x} \cdot \mathbf{y}^T\right)$$
with $\mathbf{x}\colon (\alpha, 1)$ and $\mathbf{y}\colon (\beta, 1)$, and we can reduce $e_1 \odot e_2$ to the other cases.
- If $e = e_1 \times e_2\colon (\alpha, \beta)$. By inductive hypothesis we have
$$e_1 \equiv \Sigma \mathbf{x}_1, \mathbf{y}_1, \mathbf{v}_1, \ldots, \mathbf{v}_k.\ \mathbf{s} \times \mathbf{x}_1 \cdot \mathbf{y}_1^T$$
$$e_2 \equiv \Sigma \mathbf{x}_2, \mathbf{y}_2, \mathbf{w}_1, \ldots, \mathbf{w}_l.\ \mathbf{t} \times \mathbf{x}_2 \cdot \mathbf{y}_2^T.$$
Note that $\mathbf{x}_1\colon (1, 1)$ and $\mathbf{y}_1\colon (1, 1)$. Therefore,
$$e \equiv \Sigma \mathbf{x}_2, \mathbf{y}_2, \mathbf{v}_1, \ldots, \mathbf{v}_k, \mathbf{w}_1, \ldots, \mathbf{w}_l, \mathbf{x}_1, \mathbf{y}_1.\ (\mathbf{s} \times \mathbf{t}) \times \mathbf{x}_2 \cdot \mathbf{y}_2^T.$$
- If $e = e_1 \cdot e_2\colon (\alpha, \beta)$. By inductive hypothesis we have
$$e_1 \equiv \Sigma \mathbf{x}_1, \mathbf{y}_1, \mathbf{v}_1, \ldots, \mathbf{v}_k.\ \mathbf{s} \times \mathbf{x}_1 \cdot \mathbf{y}_1^T$$
$$e_2 \equiv \Sigma \mathbf{x}_2, \mathbf{y}_2, \mathbf{w}_1, \ldots, \mathbf{w}_l.\ \mathbf{t} \times \mathbf{x}_2 \cdot \mathbf{y}_2^T.$$
Note that in this case, $\mathbf{x}_1\colon (\alpha, 1)$, $\mathbf{y}_1\colon (\gamma, 1)$, $\mathbf{x}_2\colon (\gamma, 1)$ and $\mathbf{y}_2\colon (\beta, 1)$. Therefore,
$$e \equiv \Sigma \mathbf{x}_1, \mathbf{y}_2, \mathbf{v}_1, \ldots, \mathbf{v}_k, \mathbf{w}_1, \ldots, \mathbf{w}_l, \mathbf{y}_1, \mathbf{x}_2.\ \left(\mathbf{s} \times \mathbf{t} \times \mathbf{y}_1^T \cdot \mathbf{x}_2\right) \times \mathbf{x}_1 \cdot \mathbf{y}_2^T.$$
- If $e = \Sigma \mathbf{v}.e_1\colon (\alpha, \beta)$. By inductive hypothesis we have
$$e_1 \equiv \Sigma \mathbf{x}, \mathbf{y}, \mathbf{v}_1, \ldots, \mathbf{v}_k.\ \mathbf{s} \times \mathbf{x} \cdot \mathbf{y}^T.$$
Therefore, $e \equiv \Sigma \mathbf{x}, \mathbf{y}, \mathbf{v}_1, \ldots, \mathbf{v}_k, \mathbf{v}.\ \mathbf{s} \times \mathbf{x} \cdot \mathbf{y}^T$. □

Appendix C. Omitted proofs from Section 6

Proof of Lemma 6.6.

Proof. We first observe:

(1) If $\mathit{free}(\varphi) = \{x\}$ then essentially ψ expresses a renaming of variable x into variable y, in the sense that for all db and all $\nu\colon \{y\}$ we have $[\![\psi]\!]_{db}(\nu) = [\![\varphi]\!]_{db}(x \mapsto \nu(y))$. In such a case, it hence suffices to take $\psi' := \varphi[x \leftrightarrow y]$, the $\mathsf{FO}_2^\wedge$ formula obtained from φ by simultaneously replacing all occurrences (free and bound) of x in φ by y, and all occurrences of y in φ by x.
(2) If $\mathit{free}(\varphi) = \{y\}$ then observe that $\psi \equiv \varphi$. In this case it hence suffices to take $\psi' := \varphi$.

It hence remains to show the result when $\mathit{free}(\varphi) = \{x, y\}$. The proof for this case is by well-founded induction on the length of $\mathsf{FO}_2^\wedge$ formulas.

- When φ is an atom, it must be an atom of the form $A(\overline{z})$ since $\mathit{free}(\varphi) = \{x, y\}$. Then ψ expresses that x and y must be bound to the same values in A, and x dropped from the resulting valuation. It hence suffices to take $\psi' := \varphi[x \to y]$, which is the atom obtained from $A(\overline{z})$ where all occurrences of x are replaced by y.
- The case where $\varphi = \exists z.\varphi'$ cannot happen, as this necessarily has only one free variable.
- When $\varphi = \varphi_1 \wedge \varphi_2$ we make a further case analysis.
  - If $\mathit{free}(\varphi_1) = \{x\}$ and $\mathit{free}(\varphi_2) = \{y\}$ then $\psi \equiv \exists x.(\varphi_1[x \leftrightarrow y] \wedge \varphi_2) \equiv \varphi_1[x \leftrightarrow y] \wedge \varphi_2$ where $\varphi_1[x \leftrightarrow y]$ is the formula obtained by simultaneously replacing all occurrences (free and bound) of x by y and all occurrences of y by x. Hence we take $\psi' := \varphi_1[x \leftrightarrow y] \wedge \varphi_2$.
  - If $\mathit{free}(\varphi_1) = \{x, y\}$ and $\mathit{free}(\varphi_2) = \{x, y\}$ then $\psi \equiv (\exists x.\varphi_1 \wedge x = y) \wedge (\exists x.\varphi_2 \wedge x = y)$. The formula ψ' is then obtained by applying the induction hypothesis on φ_1 and φ_2, and taking the conjunction of the resulting formulas.
  - If $\mathit{free}(\varphi_1) = \{x, y\}$ and $\mathit{free}(\varphi_2) = \{y\}$ then $\psi \equiv (\exists x.\varphi_1 \wedge x = y) \wedge \varphi_2$. The formula ψ' is obtained by applying the induction hypothesis on φ_1 and taking the conjunction of the result with φ_2.
  - If $\mathit{free}(\varphi_1) = \{x, y\}$ and $\mathit{free}(\varphi_2) = \{x\}$ then we observe that

    $$\psi \equiv \exists x.\, ((\exists y.\varphi_1 \wedge x = y) \wedge \varphi_2 \wedge x = y)\,.$$

    The latter formula first selects from φ_1 all valuations where $x = y$, projects the result on x and then multiplies this result with φ_2. By induction hypothesis there exists $\psi_1' \in \mathsf{FO}_2^\wedge$ equivalent to $\exists y.(\varphi_1 \wedge x = y)$. Let $\psi_2' := \psi_1' \wedge \varphi$. Then $\psi \equiv \exists x.(\psi_1' \wedge x = y)$. Note that ψ_1' has only one free variable, x, and hence we can apply the reasoning of (1) above to obtain our desired formula ψ.
  - If $\mathit{free}(\varphi_1) = \{x, y\}$ and $\mathit{free}(\varphi_2) = \emptyset$ then $\psi \equiv (\exists x.\varphi_1 \wedge x = y) \wedge \varphi_2$. The formula ψ' is obtained by applying the induction hypothesis on φ_1 and taking the conjunction of the result with φ_2.
  - All other cases are symmetrical variants of those already seen. Note that the cases where $\mathit{free}(\varphi_1) \cup \mathit{free}(\varphi_2) \neq \{x, y\}$ cannot occur. □

Proof of Lemma 6.7.

Proof. The proof is by induction on ψ. We assume w.l.o.g. that the only variables that occur in ψ are $\{x, y\}$. The case where ψ is an atom is immediate, and the case where ψ is a conjunction follows immediately from the induction hypothesis. Hence, assume ψ is of the form $\psi = \exists x.\psi_1$. (The case $\exists y.\psi_1$ is analogous.) By induction hypothesis, there exists ψ_1' equivalent to ψ_1 in which equality atoms only occur at the top level. If no equality atom occurs in ψ_1' then it clearly suffices to take $\psi' = \exists x.\psi_1'$. Hence, assume that some equality atom occurs in ψ_1'. We may assume w.l.o.g. that only the atom $x = y$ occurs; atoms of the form $x = x$ or $y = y$ are always valid and hence can be easily be removed. Also, because x and y are the only variables ever used, there are no other possibilities. Then, ψ_1' is of the form $\varphi \wedge x = y$ with φ an $\mathsf{FO}_2^\wedge$ formula. The result then follows by Lemma 6.6. □

Appendix D. Omitted proofs from Section 8

Proof of Lemma 8.10.

Proof. The case when $\varphi \in \mathsf{simple\text{-}FO}_2^\wedge$ is entirely analogous to the proof of Lemma 6.6 with fewer cases involved in the conjunction, since only conjunction between formulas with the same free variables is allowed.

Hence we focus when $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_k$ with $\varphi_i \in \mathsf{simple\text{-}FO}_2^\wedge$ for every i. Consider the set $F = \{\mathit{free}(\varphi_i) \mid 1 \leq i \leq k\}$. Note that $F \neq \emptyset$. Furthermore, because ϕ is a hierarchical conjunction, $\{\{x,y\},\{x\},\{y\}\} \not\subseteq F$. There are hence $2^{2^2}\text{-}3 = 13$ possibilities for F. We construct e by case analysis on F. We only illustrate three important cases; the other cases can be easily derived from the ones presented next. We adopt the following notation: for each $S \in F$ the formula φ_S denotes the conjunction of all φ_i with $\mathit{free}(\varphi_i) = S$.

- Assume $F = \{\{x\},\{y\},\emptyset\}$. Then $\varphi \equiv \varphi_x \wedge \varphi_y \wedge \varphi_\emptyset$. To express $\exists x.\varphi \wedge x = y$ it suffices to take $\psi' \coloneqq \varphi_x[x \leftrightarrow y] \wedge \varphi_y \wedge \varphi_\emptyset$, where $\varphi_y[x \leftrightarrow y]$ is the formula obtained by simultaneously replacing all occurrences (free and bound) of x by y and all occurrences of y by x.
- Assume $F = \{\{x,y\},\{y\},\emptyset\}$. Then $\varphi \equiv \varphi_{x,y} \wedge \varphi_y \wedge \varphi_\emptyset$. To express $\exists x.\varphi \wedge x = y$ it suffices to take $\psi' \coloneqq (\exists x.\varphi_{x,y} \wedge x = y) \wedge \varphi_y \wedge \varphi_\emptyset$, where $\exists x.\varphi_{x,y} \wedge x = y$ can be expressed in $\mathsf{h\text{-}FO}_2^\wedge$ since $\varphi_{x,y} \in \mathsf{simple\text{-}FO}_2^\wedge$.
- Assume $F = \{\{x,y\},\{x\},\emptyset\}$. Then $\varphi \equiv \varphi_{x,y} \wedge \varphi_x \wedge \varphi_\emptyset$. To express $\exists x.\varphi \wedge x = y$ it suffices to take $\psi' \coloneqq \exists x.\,((\exists y.\varphi_{x,y} \wedge x = y) \wedge \varphi_x \wedge \varphi_\emptyset)$, where the latter formula first selects from $\varphi_{x,y}$ all valuations where $x = y$, projects the result on x and then multiplies this result with φ_x (and $\varphi_\emptyset$). Note that here we also rely on the fact that $\exists y.\varphi_{x,y} \wedge x = y$ can be expressed in $\mathsf{h\text{-}FO}_2^\wedge$ since $\varphi_{x,y} \in \mathsf{simple\text{-}FO}_2^\wedge$. □

Proof of Lemma 8.11.

Proof. For the proof, assume w.l.o.g. that $z = y$. Let $\psi = \exists x.\varphi \wedge x = y$. Then for all $\nu\colon \{z\}$ and db we have $[\![\psi]\!]_{db}(\nu) = [\![\varphi \wedge x = y]\!]_{db}(x \mapsto \nu(z), y \mapsto \nu(z))$. It hence suffices to show that we can express $\psi(z)$ in $\mathsf{h\text{-}FO}_2^\wedge$. The result is then implied by Lemma 8.10. □

Appendix E. Related Work in Detail

E.1. **On treewidth and finite-variable logics.** Geerts and Reutter [GR22] 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. In essence, TL is a form of FO^+ evaluated over K-relations.

Geerts and Reutter [GR22] do take free variables into account when defining treewidth. Specifically, a formula with ℓ free variables has treewidth at least ℓ. Therefore, the treewidth of the following free-connex conjunctive query,

$$\varphi(x,y) = \exists z,u,v.R(x,y),R(x,z),R(z,u),R(z,v)$$

has treewidth at least 2. For this query, therefore, the result by Geerts and Reutter only implies expressibility in $\mathsf{FO}_3^\wedge$, not $\mathsf{FO}_2^\wedge$ as we show in this paper.