On matrices and K-relations

Robert Brijder, Marc Gyssens, and Jan Van den Bussche

Hasselt University, Data Science Institute, Martelarenlaan 42, 3500 Hasselt, Belgium

Abstract. We show that the matrix query language MATLANG corresponds to a natural fragment of the positive relational algebra on K-relations. The fragment is defined by introducing a composition operator and restricting K-relation arities to two. We then proceed to show that MATLANG can express all matrix queries expressible in the positive relational algebra on K-relations, when intermediate arities are restricted to three. Thus we offer an analogue, in a model with numerical data, to the situation in classical logic, where the algebra of binary relations is equivalent to first-order logic with three variables.

Keywords: expressive power, provenance semirings, annotated relations, data science

1 Introduction

Motivated by large-scale data science, there is recent interest in supporting linear algebra operations, such as matrix multiplication, in database systems. This has prompted investigations comparing the expressive power of common matrix operations with the operations on relations provided by the relational algebra and SQL [6, 7, 10, 2].

For *boolean* matrices, the connection between matrices and relations is very natural and well known. An $m \times n$ boolean matrix A can be viewed as a binary relation $R \subseteq \{1, \dots, m\} \times \{1, \dots, n\}$, where R consists of those pairs (i, j) for which $A_{i,j} = 1$. Boolean matrix multiplication then amounts to composition of binary relations. Composition is the central operation in the *algebra of binary relations* [16, 13, 15]. Besides composition, this algebra has operations such as converse, which corresponds to transposition of a boolean matrix; union and complement, which correspond to disjunction and negation of boolean matrices; and the empty and identity relations, which correspond to the zero and identity matrices.

A common theme in research in the foundations of databases is the expressive power of query languages [1]. When we employ a query language, we would like to understand as well as possible what we can do with it. Of this kind is the classical Codd theorem, stating the equivalence between the standard relational algebra and first-order logic. Likewise, for the algebra of binary relations, a classical result [17] is that it has the same expressive power as the formulas with two free variables in FO(3), the three-variable fragment of first-order logic. In this sense, we understand quite well the expressive power of a natural set of operations on boolean matrices.

What can now be said in this regard about more general matrices, with entries that are not just boolean values? An $m \times n$ matrix with entries in some semiring K is essentially a mapping from $\{1, \dots, m\} \times \{1, \dots, n\}$ to K. This perfectly fits the data model of *K-relations* introduced by Green, Garvounarakis and Tannen [5]. In general, consider an infinite domain **dom** and a supply of attributes. In a database instance, we assign to each attribute a range of values, in the form of a finite subset of **dom**. Attributes can be declared to be compatible; compatible attributes have the same range. A relation schema S is a finite set of attributes. Tuples over S are mappings that assign to each attribute a value of the appropriate range. Now a K-relation over S is a mapping that assigns to each tuple over S an element of K.

So, an $m \times n$ matrix X can be seen as a K-relation over two attributes A and B where the range of A is $\{1, \dots, m\}$ and the range of B is $\{1, \dots, n\}$. We can assume an order on all attributes and choose $A < B$ so that we know which values are row indices and which are column indices. Then an $n \times k$ matrix Y is modeled using attributes $C < D$ where we choose C and B compatible, to reflect that the number of columns of matrix X equals the number of rows of matrix Y. We can view vectors as K-relations over a single attribute, and scalars as K-relations over the empty schema. In general, a K-relation of arity r is essentially an r-dimensional tensor (multidimensional array). (Because we need not necessarily assume an order on **dom**, the tensor is unordered.)

Green et al. defined a generalization of the positive relation algebra working on K-relations, which we denote here by ARA.[1] When we restrict ARA to arities of at most three, which we denote by ARA(3), we obtain an analogue to FO(3) mentioned above. So, ARA provides a suitable scenario to reinvestigate, in a data model with numerical values, the equivalence between the algebra of binary relations and FO(3). In this paper we make the following contributions.

1. We define a suitable generalization, to K-relations, of the composition operation of classical binary relations. When we add this composition operator to ARA, but restrict arities to at most two, we obtain a natural query language for matrices. We refer to this language here as "ARA(2) plus composition".
2. We show that ARA(2) plus composition actually coincides with the matrix query language MATLANG, introduced by two of the present authors with Geerts and Weerwag [2] in an attempt to formalize the set of common matrix operations found in numerical software packages.
3. We show that a matrix query is expressible in ARA(3) if and only if it is expressible in MATLANG, thus providing an analogue to the classical result about FO(3) and the algebra of binary relations. More generally, for any arity r, we show that an r-ary query over r-ary K-relations is expressible in ARA($r+1$) if and only if it is expressible in ARA(r) plus composition. For this

[1] ARA stands for Annotated-Relation Algebra, as the elements from K that a K-relation assigns to its tuples are usually viewed as annotations.

result, we need the assumption that K is commutative. We stress that the proof is not a trivial adaptation of the proof of the classical result, because we can no longer rely on familiar classical properties like idempotence of union and join.

ARA has been a very influential vehicle for data provenance.[2] The elements from K are typically viewed as annotations, or as identifiers, and the semantics of ARA operations was originally designed to show how these annotations are propagated in the results of data manipulations. Other applications, apart from provenance, have been identified from the outset, such as security levels, or probabilities [5]. By doing the present work, we have understood that ARA can moreover serve as a fully-fledged query language for tensors (multidimensional arrays), and matrices in particular. This viewpoint is backed by the recent interest in processing Functional Aggregate Queries (FAQ [11, 12], also known as AJAR [8]). Indeed, FAQ and AJAR correspond to the project-join fragment of ARA, without self-joins.

This paper is further organized as follows. Section 2 recalls the data model of K-relations and the associated query language ARA. Section 3 presents the result on ARA$(r+1)$ and ARA(r) plus composition. Section 4 relates ARA(2) plus composition to MATLANG. Section 5 draws conclusions, discusses related work, and proposes directions for further research.

2 Annotated-Relation Algebra

By *function* we will always mean a total function. For a function $f : X \to Y$ and $Z \subseteq X$, the *restriction* of f to Z, denoted by $f|_Z$, is the function $Z \to Y$ where $f|_Z(x) = f(x)$ for all $x \in Z$.

Recall that a *semiring* K is a set equipped with two binary operations, addition $(+)$ and multiplication $(*)$, such that (1) addition is associative, commutative, and has an identity element 0; (2) multiplication is associative, has an identity element 1, and has 0 as an annihilating element; and (3) multiplication distributes over addition. A semiring is called *commutative* when multiplication is commutative. We fix a semiring K.

Remark 1. We will explicitly indicate where we assume commutativity of K.

From the outset, we also fix countable infinite sets **rel**, **att**, and **dom**, the elements of which are called *relation names*, *attributes*, and *domain elements*, respectively. We assume an equivalence relation $\sim$ on **att** that partitions **att** into an infinite number of equivalence classes that are each infinite. When $A \sim B$, we say that A and B are *compatible*. Intuitively, $A \sim B$ will mean that A and B have the same set of domain values. A function $f : X \to Y$ with X and Y sets of attributes is called *compatible* if $f(A) \sim A$ for all $A \in X$.

A *relation schema* is a finite subset of **att**. A *database schema* is a function $\mathcal{S}$ on a finite set N of relation names, assigning a relation schema $\mathcal{S}(R)$ to each

[2] The paper [5] received the PODS 2017 test-of-time award.

$R \in N$. The *arity* of a relation name R is the cardinality $|\mathcal{S}(R)|$ of its schema. The *arity* of $\mathcal{S}$ is the largest arity among relation names $R \in N$.

We now recursively define the expressions of the *Annotated-Relation Algebra*, abbreviated by ARA. At the same time we assign a relation schema to each ARA expression by extending $\mathcal{S}$ from relation names to arbitrary ARA expressions. An ARA *expression* e over a database schema $\mathcal{S}$ is one of the following:

Relation name a relation name R of $\mathcal{S}$;
One the one operation $\mathbf{1}(e')$, where e' is an ARA expression, and $\mathcal{S}(e) := \mathcal{S}(e')$;
Union the union $e_1 \cup e_2$, where e_1 and e_2 are ARA expressions with $\mathcal{S}(e_1) = \mathcal{S}(e_2)$, and $\mathcal{S}(e) := \mathcal{S}(e_1)$;
Projection the projection $\pi_Y(e')$, where e' is an ARA expression and $Y \subseteq \mathcal{S}(e')$, and $\mathcal{S}(e) := Y$;
Selection the selection $\sigma_Y(e')$, where e' is an ARA expression, $Y \subseteq \mathcal{S}(e')$, the elements of Y are mutually compatible, and $\mathcal{S}(e) := \mathcal{S}(e')$;
Renaming the renaming $\rho_\varphi(e')$, where e' is an ARA expression and $\varphi : \mathcal{S}(e') \to Y$ a compatible one-to-one correspondence with $Y \subseteq \mathbf{att}$, and $\mathcal{S}(e) := Y$; or
Join the join $e_1 \bowtie e_2$, where e_1 and e_2 are ARA expressions, and $\mathcal{S}(e) := \mathcal{S}(e_1) \cup \mathcal{S}(e_2)$.

A *domain assignment* is a function $D : \mathbf{att} \to \mathcal{D}$, where $\mathcal{D}$ is a set of nonempty finite subsets of $\mathbf{dom}$, such that $A \sim B$ implies $D(A) = D(B)$. Let X be a relation schema. A *tuple* over X with respect to D is a function $t : X \to \mathbf{dom}$ such that $t(A) \in D(A)$ for all $A \in X$. We denote by $\mathcal{T}_D(X)$ the set of tuples over X with respect to D. Note that $\mathcal{T}_D(X)$ is finite. A *relation* r over X with respect to D is a function $r : \mathcal{T}_D(X) \to K$. So a relation annotates every tuple over X with respect to D with a value from K. If $\mathcal{S}$ is a database schema, then an *instance* $\mathcal{I}$ *of* $\mathcal{S}$ *with respect to* D is a function that assigns to every relation name R of $\mathcal{S}$ a relation $\mathcal{I}(R) : \mathcal{T}_D(\mathcal{S}(R)) \to K$.

Remark 2. In practice, a domain assignment need only be defined on the attributes that are used in the database schema (and on attributes compatible to these attributes). Thus, it can be finitely specified. While here we have chosen to keep the notion of domain assignment and instance separate, it may perhaps be more natural to think of the domain assignment as being part of the instance.

$\mathcal{I}(\text{no\_courses}) =$

student	dptm	K
Alice	CS	5
Alice	Math	2
Alice	Bio	0
Bob	CS	2
Bob	Math	1
Bob	Bio	3

$\mathcal{I}(\text{course\_fee}) =$

dptm	K
CS	300
Math	250
Bio	330

Fig. 1. Example of a database instance.

Example 1. Let us record for a university both the number of courses each student takes in each department and the course fee for each department. Let K be the set of integers and let $\mathcal{S}$ be a database schema on $\{\text{no\_courses}, \text{course\_fee}\}$ with $\mathcal{S}(\text{no\_courses}) = \{\text{student}, \text{dptm}\}$ and $\mathcal{S}(\text{course\_fee}) = \{\text{dptm}\}$. Let D be a domain assignment with $D(\text{student}) = \{\text{Alice}, \text{Bob}\}$ and $D(\text{dptm}) = \{\text{CS}, \text{Math}, \text{Bio}\}$. A database instance $\mathcal{I}$ of $\mathcal{S}$ with respect to D is shown in Figure 1.

We now define the relation $\mathbf{1}_X$, as well as the generalizations of the classical operations from the positive relational algebra to work on K-relations.

One For every relation schema X, we define $\mathbf{1}_X : \mathcal{T}_D(X) \to K$ as $\mathbf{1}_X(t) = 1$.

Union Let $r_1, r_2 : \mathcal{T}_D(X) \to K$. Define $r_1 \cup r_2 : \mathcal{T}_D(X) \to K$ as $(r_1 \cup r_2)(t) = r_1(t) + r_2(t)$.

Projection Let $r : \mathcal{T}_D(X) \to K$ and $Y \subseteq X$. Define $\pi_Y(r) : \mathcal{T}_D(Y) \to K$ as

$$(\pi_Y(r))(t) = \sum_{\substack{t' \in \mathcal{T}_D(X),\\ t'|_Y = t}} r(t').$$

Selection Let $r : \mathcal{T}_D(X) \to K$ and $Y \subseteq X$ where the elements of Y are mutually compatible. Define $\sigma_Y(r) : \mathcal{T}_D(X) \to K$ such that

$$(\sigma_Y(r))(t) = \begin{cases} r(t) & \text{if } t(A) = t(B) \text{ for all } A, B \in Y; \\ 0 & \text{otherwise.} \end{cases}$$

Renaming Let $r : \mathcal{T}_D(X) \to K$ and $\varphi : X \to Y$ a compatible one-to-one correspondence. We define $\rho_\varphi(r) : \mathcal{T}_D(Y) \to K$ as $\rho_\varphi(r)(t) = r(t \circ \varphi)$.

Join Let $r_1 : \mathcal{T}_D(X_1) \to K$ and $r_2 : \mathcal{T}_D(X_2) \to K$. Define $r_1 \bowtie r_2 : \mathcal{T}_D(X_1 \cup X_2) \to K$ as $(r_1 \bowtie r_2)(t) = r_1(t|_{X_1}) * r_2(t|_{X_2})$.

The above operations provide semantics for ARA in a natural manner. Formally, let $\mathcal{S}$ be a database schema, let e be an ARA expression over $\mathcal{S}$, and let $\mathcal{I}$ be an instance of $\mathcal{S}$. The *output* relation $e(\mathcal{I})$ of e under $\mathcal{I}$ is defined as follows. If $e = R$ with R a relation name of $\mathcal{S}$, then $e(\mathcal{I}) := \mathcal{I}(R)$. If $e = \mathbf{1}(e')$, then $e(\mathcal{I}) := \mathbf{1}_{\mathcal{S}(e')}$. If $e = e_1 \cup e_2$, then $e(\mathcal{I}) := e_1(\mathcal{I}) \cup e_2(\mathcal{I})$. If $e = \pi_X(e')$, then $e(\mathcal{I}) := \pi_X(e'(\mathcal{I}))$. If $e = \sigma_Y(e')$, then $e(\mathcal{I}) := \sigma_Y(e'(\mathcal{I}))$. If $e = \rho_\varphi(e')$, then $e(\mathcal{I}) := \rho_\varphi(e'(\mathcal{I}))$. Finally, if $e = e_1 \bowtie e_2$, then $e(\mathcal{I}) := e_1(\mathcal{I}) \bowtie e_2(\mathcal{I})$.

Remark 3. The language ARA is a slight variation of the K-annotated relational algebra as originally defined by Green et al. [5] to better suit our purposes.

First of all, the original definition does not have a domain assignment $D : \mathbf{att} \to \mathcal{D}$ but instead a single domain common to all attributes (and it therefore also does not have a compatibility relation $\sim$). As such, the original definition corresponds to the case where database schemas and ARA expressions use only mutually compatible attributes. We need our more general setting when we compare ARA to MATLANG in Section 4.

Also, here, we focus on equality selections, while the original paper does not fix the allowed selection predicates. Finally, the original definition assumes zero-relations $\mathbf{0}_X$, while we instead use one-relations $\mathbf{1}_X$.

The following observations, to the effect that some (but not all) classical relational-algebra equivalences carry over to the K-annotated setting, were originally made by Green et al.

Proposition 1 ([5, Proposition 3.4]). *The following properties and equivalences hold, where, for each given equivalence, we assume that the left-hand side is well defined.*

- *Union is associative and commutative.*
- *Join is associative and distributive over union, i.e.,* $(r_1 \cup r_2) \bowtie r_3 = (r_1 \bowtie r_3) \cup (r_2 \bowtie r_3)$.
- *Any two selections commute.*
- *Projection and selection commute when projection retains the attributes on which selection takes place.*
- *Projection distributes over union, i.e.,* $\pi_Y(r_1 \cup r_2) = \pi_Y(r_1) \cup \pi_Y(r_2)$.
- *Selection distributes over union, i.e.,* $\sigma_Y(r_1 \cup r_2) = \sigma_Y(r_1) \cup \sigma_Y(r_2)$.
- *We have* $\sigma_Y(r_1) \bowtie r_2 = \sigma_Y(r_1 \bowtie r_2)$ *and* $r_1 \bowtie \sigma_Y(r_2) = \sigma_Y(r_1 \bowtie r_2)$.
- *If K is commutative, then join is commutative.*

Note that idempotence of union and of join, i.e., $r \bowtie r = r \cup r = r$, which holds for the classical relational algebra, does *not* in general hold for ARA.

We supplement Proposition 1 with the following easy-to-verify properties.

Lemma 1. *Let* $r_1 : \mathcal{T}_D(X_1) \to K$ *and* $r_2 : \mathcal{T}_D(X_2) \to K$.

- *If* $X_1 \cap X_2 \subseteq X \subseteq X_1 \cup X_2$, *then* $\pi_X(r_1 \bowtie r_2) = \pi_{X \cap X_1}(r_1) \bowtie \pi_{X \cap X_2}(r_2)$.
- *If* $Y_1, Y_2 \subseteq X_1$ *where* $Y_1 \cap Y_2 \neq \emptyset$ *and the attributes of* Y_1 *and of* Y_2 *are mutually compatible, then* $\sigma_{Y_2}(\sigma_{Y_1}(r_1)) = \sigma_{Y_1 \cup Y_2}(r_1)$.
- *If* $\varphi : X_1 \cup X_2 \to X$ *is a compatible one-to-one correspondence, then* $\rho_\varphi(r_1 \bowtie r_2) = \rho_{\varphi|_{X_1}}(r_1) \bowtie \rho_{\varphi|_{X_2}}(r_2)$. *If moreover* $X_1 = X_2$, *then* $\rho_\varphi(r_1 \cup r_2) = \rho_\varphi(r_1) \cup \rho_\varphi(r_2)$.
- *If* $Y \subseteq X_1$ *and* $\varphi : X_1 \to X$ *is a compatible one-to-one correspondence, then* $\rho_\varphi(\sigma_Y(r_1)) = \sigma_{\varphi(Y)}(\rho_\varphi(r_1))$, *where* $\varphi(Y) = \{\varphi(y) \mid y \in Y\}$.

We also use the operation of projecting away an attribute, i.e., $\hat{\pi}_A(e) := \pi_{\mathcal{S}(e)\setminus\{A\}}(e)$ if $A \in \mathcal{S}(e)$. Note that conversely, $\pi_X(e) = (\hat{\pi}_{A_m} \cdots \hat{\pi}_{A_1})(e)$ where $X = \mathcal{S}(e) \setminus \{A_1, \dots, A_m\}$ and the A_i's are mutually distinct. Projecting away, allowing one to deal with one attribute at a time, is sometimes notationally more convenient.

3 Composition and equivalence

In this section we define an operation called k-composition and show that augmenting ARA by composition allows one to reduce the required arity of the relations that are computed in subexpressions. The intuition is to provide a generalization of classical composition of two binary relations to annotated relations, so that we can compose up to k relations of arity up to k. Specifically, the classical composition of a binary relation r with a binary relation s amounts to viewing these relations as relations over schemas $\{B, A\}$ and $\{A, C\}$, respectively, and performing $\hat{\pi}_A(r \bowtie s)$. Thus we arrive at the following generalization.

Definition 1. *Let k be a nonnegative integer and let $l \in \{1, \ldots, k\}$. Let $r_i : \mathcal{T}_D(X_i) \to K$ for $i \in \{1, \ldots, l\}$, let $X = X_1 \cup \cdots \cup X_l$, and let $A \in X_1 \cap \cdots \cap X_l$.*

*Define the k-*composition *$\zeta_{A,k}(r_1, \ldots, r_l) : \mathcal{T}_D(X \setminus \{A\}) \to K$ as*

$$(\zeta_{A,k}(r_1, \ldots, r_l))(t) = (\hat{\pi}_A(r_1 \bowtie \cdots \bowtie r_l))(t)$$

for all $t \in \mathcal{T}_D(X \setminus \{A\})$.

Note that $\zeta_{A,k}$ takes at most k arguments. We emphasize that $\zeta_{A,k}$ is defined as a new operator (albeit one that can be defined by an ARA expression) and not as a shorthand for an ARA expression.

We denote by $\mathsf{ARA} + \zeta_k$ the language obtained by extending ARA with k-composition. Consequently, if $e_1, \ldots, e_l$ are $\mathsf{ARA} + \zeta_k$ expressions with $l \leq k$ and $A \in \mathcal{S}(e_1) \cap \cdots \cap \mathcal{S}(e_l)$, then $e = \zeta_{A,k}(e_1, \ldots, e_l)$ is an $\mathsf{ARA} + \zeta_k$ expression. Also, we let $\mathcal{S}(e) := (\mathcal{S}(e_1) \cup \cdots \cup \mathcal{S}(e_l)) \setminus \{A\}$.

Let k be a nonnegative integer. We denote by $\mathsf{ARA}(k)$ the fragment of ARA in which the database schemas are restricted to arity at most k and the relation schema of each subexpression is of cardinality at most k. In particular, join $e_1 \bowtie e_2$ is only allowed if $|\mathcal{S}(e_1 \bowtie e_2)| \leq k$. The fragment $(\mathsf{ARA} + \zeta_k)(k)$ is defined similarly.

From Definition 1 it is apparent that $(\mathsf{ARA} + \zeta_k)(k)$ is subsumed by $\mathsf{ARA}(k + 1)$. One of our main results (Corollary 1) provides the converse inclusion, when the database schemas and outputs are restricted to arity at most k. To this end, we establish a normal form for ARA expressions.

We use the following terminology. Let $\mathcal{F}$ be any family of expressions. A *selection of $\mathcal{F}$-expressions* is an expression of the form $\sigma_{Y_n} \cdots \sigma_{Y_1}(f)$, where f is an $\mathcal{F}$-expression and $n \geq 0$. Note the slight abuse of terminology as we allow multiple selection operations. Also, when we say that e is a *union of $\mathcal{F}$-expressions* or a *join of $\mathcal{F}$-expressions*, we allow e to be just a single expression in $\mathcal{F}$ (so union and join may be skipped).

We are now ready to formulate a main result of this paper. This result is inspired by the classic equivalence of FO(3) and the algebra of binary relations [17]. A compact proof of this classical result is given by Marx and Venema [14, Theorem 3.4.5, Claim 2], and a self-contained exposition is also available [3].

Two ARA expressions e_1 and e_2 over the same database schema are called *equivalent*, naturally, if they yield the same output relation for every domain assignment and every database instance respecting that domain assignment.

Theorem 1. *Let $\mathcal{S}$ be a database schema of arity at most k and assume that K is commutative. Every $\mathsf{ARA}(k + 1)$ expression over $\mathcal{S}$ is equivalent to a union of selections of joins of $(\mathsf{ARA} + \zeta_k)(k)$ expressions over $\mathcal{S}$.*

The proof of Theorem 1 uses Proposition 1, Lemma 1, and the following technical lemma to effectively construct the expression in the form given by Theorem 1. This effective construction is illustrated in Example 2.

Lemma 2. *Let $r_1, \dots, r_n$ be relations with relation schemas $X_1, \dots, X_n$, respectively, and with respect to a domain assignment D. Assume that $A, B \in X_1 \cup \dots \cup X_n$ are distinct and compatible. Define, for $i \in \{1, \dots, n\}$,*

$$r'_i := \begin{cases} r_i & \text{if } A \notin X_i; \\ \rho_{A \to B}(r_i) & \text{if } A \in X_i, B \notin X_i; \\ \hat{\pi}_A(\sigma_{\{A,B\}}(r_i)) & \text{if } A, B \in X_i, \end{cases}$$

where $A \to B$ denotes the one-to-one correspondence from X_i to $(X_i \setminus \{A\}) \cup \{B\}$ that assigns A to B and keeps the remaining attributes fixed. Then

$$\hat{\pi}_A(\sigma_{\{A,B\}}(r_1 \bowtie \dots \bowtie r_n)) = r'_1 \bowtie \dots \bowtie r'_n.$$

Example 2. Assume that K is commutative and consider the $\mathsf{ARA}(3)$ expression $e = \pi_{\{B,C\}}(\sigma_{\{B,C\}}(R \bowtie R \bowtie S \bowtie T \bowtie \rho_\varphi(T)) \cup \sigma_{\{A,B\}}(R \bowtie S \bowtie T))$, where $\mathcal{S}(R) = \{A, B\}$, $\mathcal{S}(S) = \{B, C\}$, $\mathcal{S}(T) = \{A, C\}$ (A, B, C are mutually distinct), and φ sends A to B and C to itself. The proof of Theorem 1 obtains an equivalent expression in normal form by using Proposition 1, Lemma 1, and Lemma 2 as follows.

$$\begin{aligned} e &= \hat{\pi}_A(\sigma_{\{B,C\}}(R \bowtie R \bowtie S \bowtie T \bowtie \rho_\varphi(T)) \cup \sigma_{\{A,B\}}(R \bowtie S \bowtie T)) \\ &\equiv \hat{\pi}_A(\sigma_{\{B,C\}}(R \bowtie R \bowtie S \bowtie T \bowtie \rho_\varphi(T))) \cup \hat{\pi}_A(\sigma_{\{A,B\}}(R \bowtie S \bowtie T)) \\ &\equiv \sigma_{\{B,C\}}(\hat{\pi}_A(R \bowtie R \bowtie S \bowtie T \bowtie \rho_\varphi(T))) \cup \hat{\pi}_A(\sigma_{\{A,B\}}(R \bowtie S \bowtie T)) \\ &\equiv \sigma_{\{B,C\}}(S \bowtie \rho_\varphi(T) \bowtie \hat{\pi}_A(R \bowtie R \bowtie T)) \cup \hat{\pi}_A(\sigma_{\{A,B\}}(R \bowtie S \bowtie T)) \\ &\equiv \sigma_{\{B,C\}}(S \bowtie \rho_\varphi(T) \bowtie \zeta_{A,2}(R \bowtie R, T)) \cup \hat{\pi}_A(\sigma_{\{A,B\}}(R \bowtie S \bowtie T)) \\ &\equiv \sigma_{\{B,C\}}(S \bowtie \rho_\varphi(T) \bowtie \zeta_{A,2}(R \bowtie R, T)) \cup \big(\hat{\pi}_A(\sigma_{\{A,B\}}(R)) \bowtie S \bowtie \rho_{A \to B}(T)\big). \end{aligned}$$

The first two equivalences follow from Proposition 1, the third equivalence follows from Lemma 1, the fourth equivalence is by the definition of $\zeta_{A,2}$, and the last equivalence is by Lemma 2. The last expression is in the normal form since the subexpressions S, $\rho_\varphi(T)$, $\zeta_{A,2}(R \bowtie R, T)$, $\hat{\pi}_A(\sigma_{\{A,B\}}(R))$, and $\rho_{A \to B}(T)$ are all $(\mathsf{ARA} + \zeta_2)(2)$ expressions.

Note that we likely cannot omit the "selections of" in the above theorem. For example, for $k = 2$ consider $\sigma_{\{A,C\}}(R \bowtie S)$ where R and S are relation names with $\mathcal{S}(R) = \{A, B\}$ and $\mathcal{S}(S) = \{B, C\}$.

Remark 4. Theorem 1 still holds if the **1** operator is omitted from the definition of ARA.

Since union, selection, and join do not decrease the number of attributes of relations, we have the following corollary to Theorem 1, which establishes the main result announced in the Introduction.

Corollary 1. *Let $\mathcal{S}$ be a database schema of arity at most k and assume that K is commutative. Every $\mathsf{ARA}(k+1)$ expression e over $\mathcal{S}$ with $|\mathcal{S}(e)| \leq k$ is equivalent to an $(\mathsf{ARA} + \zeta_k)(k)$ expression over $\mathcal{S}$.*

Remark 5. We remark that transforming an expression into the normal form of Theorem 1 may lead to an exponential increase in expression length. The reason is that the proof uses distributivity of join over union. Indeed, each time we replace an expression of the form $(e_1 \cup e_2) \bowtie e_3$ by $(e_1 \bowtie e_3) \cup (e_2 \bowtie e_3)$ there is a duplication of e_3. The proof of the classic translation of FO(3) to the algebra of binary relations also induces an exponential increase of expression length for similar reasons. A proof that this blowup is unavoidable remains open, both for our result and for the classical result (to the best of our knowledge).

3.1 Connection with FO(k)

The connection between $\mathsf{ARA}(k)$ and $\mathrm{FO}(k)$, to which we have hinted several times already, can be made explicit as follows.

Let K be the Boolean semiring. Consider a database schema $\mathcal{S}$ of arity at most k. Let D be a domain assignment such that $D(A) = D(B)$ for all attributes A and B. In other words, D just fixes a single nonempty finite subset of **dom**. A tuple (D, I), with I an instance over $\mathcal{S}$, is a classical relational structure over $\mathcal{S}$.

We can consider $\mathrm{FO}(k)$ formulas as first-order logic formulas (also known as relational-calculus formulas [1]) over $\mathcal{S}$ that use only k distinct variables. It is then an easy exercise to see that every $\mathrm{FO}(k)$ formula that does not use negation or universal quantification, and has $k' \leq k$ free variables can be translated to an equivalent $\mathsf{ARA}(k)$ expression. The converse translation is also possible.

4 Matrices

In this section we show that $(\mathsf{ARA} + \zeta_2)(2)$ is equivalent to a natural version of $\mathsf{MATLANG}$ [2]. As a consequence of Corollary 1, we then obtain that also $\mathsf{ARA}(3)$, with database schemas and output relations restricted to arity at most 2, is equivalent to $\mathsf{MATLANG}$. We begin by recalling the definition of this language.

4.1 MATLANG

Let us fix the countable infinite sets **matvar** and **size**, where the latter has a distinguished element $1 \in$ **size**. The elements of **matvar** are called *matrix variables* and the elements of **size** are called *size symbols*.

A *matrix schema* is a function $\mathcal{S} : V \to \mathbf{size} \times \mathbf{size}$ with $V \subseteq \mathbf{matvar}$ both finite and nonempty. We write $(\alpha, \beta) \in \mathbf{size} \times \mathbf{size}$ also as $\alpha \times \beta$.

$\mathsf{MATLANG}$ expressions are recursively defined as follows. At the same time we assign a matrix schema to each $\mathsf{MATLANG}$ expression by extending $\mathcal{S}$ from matrix variables to arbitrary $\mathsf{MATLANG}$ expressions.

A $\mathsf{MATLANG}$ *expression* e over a matrix schema $\mathcal{S}$ is one of the following:

Variable a matrix variable M of $\mathcal{S}$;
Transposition a transposition $(e')^T$, where e' is a $\mathsf{MATLANG}$ expression, and $\mathcal{S}(e) := \beta \times \alpha$ if $\mathcal{S}(e') = \alpha \times \beta$;

One-vector a one-vector $\mathbf{1}(e')$, where e' is a MATLANG expression, and $\mathcal{S}(e) := \alpha \times 1$ if $\mathcal{S}(e') = \alpha \times \beta$;

Diagonalization a diagonalization $\mathrm{diag}(e')$, where e' is a MATLANG expression with $\mathcal{S}(e') = \alpha \times 1$, and $\mathcal{S}(e) := \alpha \times \alpha$;

Multiplication a multiplication $e_1 \cdot e_2$, where e_1 and e_2 are MATLANG expressions with $\mathcal{S}(e_1) = \alpha \times \beta$ and $\mathcal{S}(e_2) = \beta \times \gamma$, and $\mathcal{S}(e) := \alpha \times \gamma$;

Addition an addition $e_1 + e_2$, where e_1 and e_2 are MATLANG expressions with $\mathcal{S}(e_1) = \mathcal{S}(e_2)$, and $\mathcal{S}(e) := \mathcal{S}(e_1)$; or

Hadamard product a Hadamard product $e_1 \circ e_2$, where e_1 and e_2 are MATLANGexpressions with $\mathcal{S}(e_1) = \mathcal{S}(e_2)$, and $\mathcal{S}(e) := \mathcal{S}(e_1)$.

A *size assignment* is a function σ that assigns to each size term a strictly positive integer with $\sigma(1) = 1$. Let $\mathcal{M}$ be the set of all matrices over K. We say that $M \in \mathcal{M}$ *conforms* to $\alpha \times \beta \in \mathbf{size} \times \mathbf{size}$ by σ if M is a $\sigma(\alpha) \times \sigma(\beta)$-matrix.

If $\mathcal{S} : V \to \mathbf{size} \times \mathbf{size}$ is a matrix schema, then an *instance of* $\mathcal{S}$ *with respect to* σ is a function $\mathcal{I} : V \to \mathcal{M}$ such that, for each $M \in V$, the matrix $\mathcal{I}(M)$ conforms to $\mathcal{S}(M)$ by σ.

Remark 6. In practice, a size assignment need only be defined on the size terms that are used in the schema. Thus, it can be finitely specified. While here we have chosen to keep the notion of size assignment and instance separate, it may perhaps be more natural to think of the size assignment as being part of the instance.

$$\mathcal{I}(\text{no\_courses}) = \begin{pmatrix} 5 & 2 & 0 \\ 2 & 1 & 3 \end{pmatrix} \qquad \mathcal{I}(\text{course\_fee}) = \begin{pmatrix} 300 \\ 250 \\ 330 \end{pmatrix}$$

Fig. 2. An example of an instance of a matrix schema.

Example 3. This example is similar to Example 1. Let K be the set of integers and let $\mathcal{S}$ be a matrix schema on $\{\text{no\_courses}, \text{course\_fee}\}$ with $\mathcal{S}(\text{no\_courses}) = \text{student} \times \text{dptm}$ and $\mathcal{S}(\text{course\_fee}) = \text{dptm} \times 1$. Let σ be a size assignment with $\sigma(\text{student}) = 2$ and $\sigma(\text{dptm}) = 3$. An instance $\mathcal{I}$ of $\mathcal{S}$ with respect to σ is shown in Figure 2.

The semantics for MATLANG is given by the following matrix operations. Let A be an $m \times n$-matrix over K. We define $\mathbf{1}(A)$ to be the $m \times 1$-matrix (i.e., column vector) with $\mathbf{1}(A)_{i,1} = 1$. If $n = 1$ (i.e., A is a column vector), then $\mathrm{diag}(A)$ is the $m \times m$-matrix with $\mathrm{diag}(A)_{i,j}$ equal to $A_{i,1}$ if $i = j$ and to 0 otherwise. If B is an $m \times n$-matrix, then $A \circ B$ denotes the Hadamard product of A and B. In other words, $(A \circ B)_{i,j} = A_{i,j} * B_{i,j}$. Matrix addition and matrix multiplication are as usual denoted by $+$ and $\cdot$, respectively.

Formally, let $\mathcal{S}$ be a matrix schema, let e be a $\mathsf{MATLANG}$ expression over $\mathcal{S}$, and let $\mathcal{I}$ be a matrix instance of $\mathcal{S}$. Then the *output* matrix $e(\mathcal{I})$ of e under $\mathcal{I}$ is defined in the obvious way, given the operations just defined. If $e = M$ with M a matrix variable of $\mathcal{S}$, then $e(\mathcal{I})$ is naturally defined to be equal to $\mathcal{I}(M)$.

Remark 7. Matrix addition and the Hadamard product are the pointwise applications of addition and product, respectively. The original definition of $\mathsf{MATLANG}$ [2] is more generally defined in terms of an arbitrary set Ω of allowed pointwise functions. So, $\mathsf{MATLANG}$ as defined above fixes Ω to $\{+, \cdot\}$. This restriction was also considered by Geerts [4] (who also allows multiplication by constant scalars, but this is not essential).

Also, the original definition of $\mathsf{MATLANG}$ fixes K to the field of complex numbers and complex transpose is considered instead of (ordinary) transpose. Of course, transpose can be expressed using complex transpose and pointwise application of conjugation.

Table 1. Symbol table for the simulations between $\mathsf{MATLANG}$ and $(\mathsf{ARA} + \zeta_2)(2)$.

Mapping	$\mathsf{MATLANG} \to \mathsf{ARA}$	$\mathsf{ARA} \to \mathsf{MATLANG}$
attributes A/size terms α	row_α, col_α	$\Psi(A)$
schemas $\mathcal{S}$	$\Gamma(\mathcal{S})$	$\Theta(\mathcal{S})$
expressions e	$\Upsilon(e)$	$\Phi(e)$
instances I, relations r/matrices M	$\mathrm{Rel}_{\mathcal{S},\sigma}(I)$, $\mathrm{Rel}_{s,\sigma}(M)$	$\mathrm{Mat}_D(I)$, $\mathrm{Mat}_D(r)$

In the following subsections we provide simulations between $\mathsf{MATLANG}$ and $(\mathsf{ARA} + \zeta_2)(2)$. The notations for the different translations that will be given are summarized in Table 1.

4.2 Simulating MATLANG in $(\mathsf{ARA} + \zeta_2)(2)$

For notational convenience, instead of fixing a one-to-one correspondence between **rel** and **matvar**, we assume that **rel** = **matvar**.

Let us now fix injective functions row : $\mathbf{size} \setminus \{1\} \to \mathbf{att}$ and col : $\mathbf{size} \setminus \{1\} \to \mathbf{att}$ such that (1) $\mathrm{row}(\alpha)$ and $\mathrm{col}(\alpha)$ are compatible for all $\alpha \in \mathbf{size} \setminus \{1\}$, and (2) the range of row is disjoint from the range of col. To reduce clutter, we also write, for $\alpha \in \mathbf{size} \setminus \{1\}$, $\mathrm{row}(\alpha)$ as row_α and $\mathrm{col}(\alpha)$ as col_α.

Let $s \in \mathbf{size} \times \mathbf{size}$. We associate to s a relation schema $\Gamma(s)$ with $|\Gamma(s)| \leq 2$ as follows.

$$\Gamma(s) := \begin{cases} \{\mathrm{row}_\alpha, \mathrm{col}_\beta\} & \text{if } s = \alpha \times \beta; \\ \{\mathrm{row}_\alpha\} & \text{if } s = \alpha \times 1; \\ \{\mathrm{col}_\beta\} & \text{if } s = 1 \times \beta; \\ \emptyset & \text{if } s = 1 \times 1, \end{cases}$$

where $\alpha \neq 1 \neq \beta$.

Let $\mathcal{S}$ be a matrix schema on a set of matrix variables V. We associate to $\mathcal{S}$ a database schema $\Gamma(\mathcal{S})$ on V as follows. For $M \in V$, we set $(\Gamma(\mathcal{S}))(M) := \Gamma(\mathcal{S}(M))$.

Let σ be a size assignment. We associate to σ a domain assignment $D(\sigma)$ where, for $\alpha \in \mathbf{size}$, $(D(\sigma))(\mathrm{row}_\alpha) := (D(\sigma))(\mathrm{col}_\alpha) := \{1, \ldots, \sigma(\alpha)\}$.

Let $M \in \mathcal{M}$ conform to $s = \alpha \times \beta$ by σ. We associate to M a relation $\mathrm{Rel}_{s,\sigma}(M) : \mathcal{T}_{D(\sigma)}(\Gamma(s)) \to K$ as follows. We have $(\mathrm{Rel}_{s,\sigma}(M))(t) := M_{i,j}$, where (1) i is equal to $t(\mathrm{row}_\alpha)$ if $\alpha \neq 1$ and equal to 1 if $\alpha = 1$; and (2) j is equal to $t(\mathrm{col}_\beta)$ if $\beta \neq 1$ and equal to 1 if $\beta = 1$.

Let $\mathcal{S} : V \to \mathbf{size} \times \mathbf{size}$ be a matrix schema and let $\mathcal{I}$ be a matrix instance of $\mathcal{S}$ with respect to σ. We associate to $\mathcal{I}$ an instance $\mathrm{Rel}_{\mathcal{S},\sigma}(\mathcal{I})$ of database schema $\Gamma(\mathcal{S})$ with respect to $D(\sigma)$ as follows. For $M \in V$, we set $(\mathrm{Rel}_{\mathcal{S},\sigma}(\mathcal{I}))(M) := \mathrm{Rel}_{\mathcal{S}(M),\sigma}(\mathcal{I}(M))$.

$\mathcal{I}(\text{no\_courses}) =$

$\mathrm{row}_{\mathrm{student}}$	$\mathrm{col}_{\mathrm{dptm}}$	K
1	1	5
1	2	2
1	3	0
2	1	2
2	2	1
2	3	3

$\mathcal{I}(\text{course\_fee}) =$

$\mathrm{row}_{\mathrm{dptm}}$	K
1	300
2	250
3	330

Fig. 3. Matrix instance from Figure 2 represented as a database instance.

Example 4. Recall $\mathcal{I}$, $\mathcal{S}$, and σ from Example 3. We have that $(\Gamma(\mathcal{S}))(\text{no\_courses}) = \{\mathrm{row}_{\mathrm{student}}, \mathrm{col}_{\mathrm{dptm}}\}$ and $(\Gamma(\mathcal{S}))(\text{course\_fee}) = \{\mathrm{row}_{\mathrm{dptm}}\}$. The database instance $\mathrm{Rel}_{\mathcal{S},\sigma}(\mathcal{I})$ is shown in Figure 3.

The next lemma shows that every $\mathsf{MATLANG}$ expression can be simulated by an $(\mathsf{ARA} + \zeta_2)(2)$ expression.

Lemma 3. *For each* $\mathsf{MATLANG}$ *expression e over a matrix schema $\mathcal{S}$, there exists an* $(\mathsf{ARA}+\zeta_2)(2)$ *expression $\Upsilon(e)$ over database schema $\Gamma(\mathcal{S})$ such that (1) $\Gamma(\mathcal{S}(e)) = (\Gamma(\mathcal{S}))(\Upsilon(e))$ and (2) for all size assignments σ and matrix instances $\mathcal{I}$ of $\mathcal{S}$ with respect to σ, we have* $\mathrm{Rel}_{\mathcal{S}(e),\sigma}(e(\mathcal{I})) = (\Upsilon(e))(\mathrm{Rel}_{\mathcal{S},\sigma}(\mathcal{I}))$.

Example 5. We continue the running example. In particular, recall $\mathcal{I}$, $\mathcal{S}$, and σ from Example 3. Consider the $\mathsf{MATLANG}$ expression $e = \text{no\_courses} \cdot \text{course\_fee}$ over $\mathcal{S}$. We have $\mathcal{S}(e) = \text{student} \times 1$ and

$$e(\mathcal{I}) = \begin{pmatrix} 2000 \\ 1840 \end{pmatrix};$$

$\mathrm{Rel}_{\mathcal{S}(e),\sigma}(e(\mathcal{I})) =$

$\mathrm{row}_{\mathrm{student}}$	K
1	2000
2	1840

.

By Lemma 3 and its proof, we have that $\mathrm{Rel}_{\mathcal{S}(e),\sigma}(e(\mathcal{I}))$ is equal to $e'(\mathrm{Rel}_{\mathcal{S},\sigma}(\mathcal{I}))$ with

$$e' = \zeta_{C,2}(\rho_{\varphi_1}(\text{no\_courses}), \rho_{\varphi_2}(\text{course\_fee})),$$

where $\varphi_1(\mathrm{col}_\gamma) = \varphi_2(\mathrm{row}_\gamma) = C \notin \{\mathrm{row}_\alpha, \mathrm{col}_\beta\}$ and φ_1 and φ_2 are the identity otherwise.

4.3 Simulating $(\mathsf{ARA} + \zeta_2)(2)$ in $\mathsf{MATLANG}$

In order to simulate $(\mathsf{ARA} + \zeta_2)(2)$ in $\mathsf{MATLANG}$, we equip **att** with some linear ordering $<$. We remark that $<$ is an ordering on attributes, not on domain elements. Only an ordering on domain elements can have an impact on the expressive power of query languages of query languages [1].

Again we assume that **rel** $=$ **matvar**. Let us fix an injective function Ψ : **att** $\to$ **size** $\setminus \{1\}$.

Let $X \subseteq \{A_1, A_2\}$ be a relation schema for some A_1 and A_2 with $A_1 < A_2$. We associate to X an element $\Theta(X) \in$ **size** $\times$ **size** as follows. We have

$$\Theta(X) := \begin{cases} \Psi(A_1) \times \Psi(A_2) & \text{if } X = \{A_1, A_2\}; \\ \Psi(A) \times 1 & \text{if } X = \{A\} \text{ for some } A; \\ 1 \times 1 & \text{if } X = \emptyset. \end{cases}$$

Let $\mathcal{S}$ a database schema on a set N of relation names of arities at most 2. We associate to $\mathcal{S}$ a matrix schema $\Theta(\mathcal{S})$ on N as follows. For $R \in N$, we set $(\Theta(\mathcal{S}))(R) := \Theta(\mathcal{S}(R))$.

Let D be a domain assignment. We associate to D a size assignment $\sigma(D)$ where, for $A \in$ **att**, $(\sigma(D))(D(A)) = |D(A)|$. If every element in the range of a domain assignment D is of the form $\{1, \dots, n\}$ for some n, then we say that D is *consecutive*.

Let D be a consecutive domain assignment. Given a relation $r : \mathcal{T}_D(X) \to K$ with $X \subseteq \{A_1, A_2\}$ and $A_1 < A_2$, we associate a matrix $\mathrm{Mat}_D(r)$ conforming to $\Theta(X)$ by $\sigma(D)$ as follows. We define $(\mathrm{Mat}_D(r))_{i,j} := r(t)$, where t is (1) the tuple with $t(A_1) = i$ and $t(A_2) = j$ if $|X| = 2$; (2) the tuple with $t(A) = i$ and $j = 1$ if $X = \{A\}$ for some A; and (3) the unique tuple of $\mathcal{T}_D(X)$ if $X = \emptyset$.

Let $\mathcal{S}$ a database schema on a set N of relation names of arities at most 2, and let $\mathcal{I}$ be a database of $\mathcal{S}$ instance with respect to D. We associate to $\mathcal{I}$ a matrix instance $\mathrm{Mat}_D(\mathcal{I})$ of $\mathrm{Mat}(\mathcal{S})$ with respect to $\sigma(D)$ as follows. For $R \in N$, we set $(\mathrm{Mat}_D(\mathcal{I}))(R) := \mathrm{Mat}_D(\mathcal{I}(R))$.

Example 6. Recall $\mathcal{I}$, $\mathcal{S}$, and D from Example 1. To reduce clutter, assume that **att** $=$ **size** $\setminus \{1\}$ and that Ψ is the identity function. Take student $<$ dptm. We have that $(\Theta(\mathcal{S}))(\text{no\_courses}) = \text{student} \times \text{dptm}$ and $(\Theta(\mathcal{S}))(\text{course\_fee}) = \text{dptm} \times 1$. Consider domain assignment D' and database instance $\mathcal{I}'$ obtained from D and $\mathcal{I}$, respectively, by replacing Alice by 1, Bob by 2, CS by 1, Math by 2, and Bio by 3. Note that D' is consecutive. The instance $\mathrm{Mat}_{D'}(\mathcal{I}')$ is shown in Figure 2.

The next lemma shows that every $(\mathsf{ARA}+\zeta_2)(2)$ expression can be simulated by a MATLANG expression.

Lemma 4. *For each* $(\mathsf{ARA}+\zeta_2)(2)$ *expression e over a database schema $\mathcal{S}$ of arity at most 2, there exists a* MATLANG *expression $\Phi(e)$ over matrix schema $\Theta(\mathcal{S})$ such that (1) $\Theta(\mathcal{S}(e)) = (\Theta(\mathcal{S}))(\Phi(e))$ and (2) for all consecutive domain assignments D and database instances $\mathcal{I}$ with respect to D, we have* $\mathrm{Mat}_D(e(\mathcal{I})) = (\Phi(e))(\mathrm{Mat}_D(\mathcal{I}))$.

Example 7. We continue the running example. In particular, recall $\mathcal{I}'$, $\mathcal{S}$, and D' from Examples 1 and 6. Consider the $(\mathsf{ARA}+\zeta_2)(2)$ expression $e = \text{no\_courses} \bowtie \text{course\_fee}$ over $\mathcal{S}$. We have $\mathcal{S}(e) = \{\text{student}, \text{dptm}\}$ and

$e(\mathcal{I}') =$

student	dptm	K
1	1	1500
1	2	500
1	3	0
2	1	600
2	2	250
2	3	990

;

$$\mathrm{Mat}_{D'}(e(\mathcal{I}')) = \begin{pmatrix} 1500 & 500 & 0 \\ 600 & 250 & 990 \end{pmatrix}.$$

By Lemma 3 and its proof, we have that $\mathrm{Mat}_{D'}(e(\mathcal{I}'))$ is equal to $e'(\mathrm{Mat}_{D'}(\mathcal{I}'))$ with $e' = \text{no\_courses} \cdot \text{diag}(\text{course\_fee})$.

4.4 Relationship with ARA(3) and complexity

Corollary 1, Lemma 3, and Lemma 4 together establish the equivalence of MATLANG with the language ARA(3) restricted to database schemas and output relations of arity at most 2.

Theorem 2. *For each* ARA(3) *expression e over a database schema $\mathcal{S}$ of arity at most 2 and with $|\mathcal{S}(e)| \leq 2$, there exists a* MATLANG *expression e' such that $\mathrm{Mat}_D(e(\mathcal{I})) = e'(\mathrm{Mat}_D(\mathcal{I}))$ for all consecutive domain assignments D and instances $\mathcal{I}$ with respect to $\mathcal{S}$ over D.*

Conversely, for each MATLANG *expression e over a matrix schema $\mathcal{S}$, there exists an* ARA(3) *expression e' such that $\mathrm{Rel}_{\mathcal{S}(e),\sigma}(e(\mathcal{I})) = e'(\mathrm{Rel}_{\mathcal{S},\sigma}(\mathcal{I}))$ for all size assignments σ and matrix instances $\mathcal{I}$ of $\mathcal{S}$ with respect to σ.*

As to complexity, we note that Υ and Φ in Lemmas 3 and 4 can be effectively constructed and of linear length (for fixed schemas; quadratic when the schema is part of the input). We conclude that the direction MATLANG $\rightarrow$ $(\mathsf{ARA}+\zeta_2)(2)$ $\rightarrow$ ARA(3) in the above result is linear. The direction ARA(3) $\rightarrow$ $(\mathsf{ARA}+\zeta_2)(2)$, however, is exponential, see Remark 5.

5 Conclusion

In related work, Yan, Tannen, and Ives consider provenance for linear algebra operators [18]. In that approach, provenance tokens represent not the matrix entries (as in our work), but the matrices themselves. Polynomial expressions (with matrix addition and matrix multiplication) are derived to show the provenance of linear algebra operations applied to these matrices.

Our result that every matrix query expressible in $\mathsf{ARA}(3)$ is also expressible in MATLANG provides a partial converse to the observation already made in the original paper [2], to the effect that MATLANG can be expressed in $\mathcal{L}_{\mathrm{Aggr}}(3)$: the relational calculus with summation and numerical functions [9], restricted to three base variables.[3] This observation was made in the extended setting of MATLANG that allows arbitrary pointwise functions (Remark 7). For the language considered here, $\mathsf{ARA}(3)$ provides a more appropriate upper bound for comparison, and $\mathsf{ARA}(3)$ is still a natural fragment of $\mathcal{L}_{\mathrm{Aggr}}(3)$.

When allowing arbitrary pointwise functions in MATLANG, we actually move beyond the positive relational algebra, as queries involving negation can be expressed. For example, applying the function $x \wedge \neg y$ pointwise to the entries of two $n \times n$ boolean matrices representing two binary relations R and S on $\{1, \ldots, n\}$, we obtain the set difference $R - S$. It is an interesting research question to explore expressibility of queries in MATLANG in this setting. For example, consider the following $\mathcal{L}_{\mathrm{Aggr}}(3)$ query on two matrices M and N:

$$\forall i \exists j \forall k \forall x (M(i,k,x) \to \exists i\, N(j,i,x))$$

Here, $M(i,k,x)$ means that $M_{i,k} = x$, and similarly for $N(j,i,x)$.

The above query, which does not even use summation, reuses the base variable i and checks whether each row of M, viewed as a set of entries, is included in some row of N, again viewed as a set of entries. We conjecture that the query is not expressible in MATLANG with arbitrary pointwise functions. Developing techniques for showing this is an interesting direction for further research.

Finally, recall that our main result Corollary 1 assumes that K is commutative. It should be investigated whether or not this result still holds in the noncommutative case.

Acknowledgments We thank Floris Geerts for inspiring discussions. Robert Brijder has been a postdoctoral fellow of the Research Foundation - Flanders (FWO). Jan Van den Bussche was partially supported by the National Natural Science Foundation of China under grant# 61972455.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Brijder, R., Geerts, F., Van den Bussche, J., Weerwag, T.: On the expressive power of query languages for matrices. In: Kimelfeld, B., Amsterdamer, Y. (eds.) Proceedings 21st International Conference on Database Theory. LIPIcs, vol. 98, pp. 10:1–10:17. Schloss Dagstuhl–Leibniz Center for Informatics (2018)
3. Van den Bussche, J.: FO^3 and the algebra of binary relations. `https://databasetheory.org/node/94`, retrieved 22 July 2019

[3] $\mathcal{L}_{\mathrm{Aggr}}$ is a two-sorted logic with base variables and numerical variables.

4. Geerts, F.: On the expressive power of linear algebra on graphs. In: Barcelo, P., Calautti, M. (eds.) Proceedings 22nd International Conference on Database Theory. LIPIcs, vol. 127, pp. 7:1–7:19. Schloss Dagstuhl–Leibniz Center for Informatics (2019)
5. Green, T., Karvounarakis, G., Tannen, V.: Provenance semirings. In: Proceedings 26th ACM Symposium on Principles of Database Systems. pp. 31–40 (2007)
6. Hutchison, D., Howe, B., Suciu, D.: LaraDB: A minimalist kernel for linear and relational algebra computation. In: Afrati, F., Sroka, J. (eds.) Proceedings 4th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond. pp. 2:1–2:10 (2017)
7. Jananthan, H., Zhou, Z., et al.: Polystore mathematics of relational algebra. In: Nie, J.Y., Obradovic, Z., Suzumura, T., et al. (eds.) Proceedings IEEE International Conference on Big Data. pp. 3180–3189. IEEE (2017)
8. Joglekar, M., Puttagunta, R., Ré, C.: AJAR: Aggregations and joins over annotated relations. In: Proceedings 35th ACM Symposium on Principles of Databases. pp. 91–106. ACM (2016)
9. Libkin, L.: Expressive power of SQL. Theor. Comput. Sci. **296**, 379–404 (2003)
10. Luo, S., Gao, Z., Gubanov, M., Perez, L., Jermaine, C.: Scalable linear algebra on a relational database system. SIGMOD Rec. **47**(1), 24–31 (2018)
11. Abo Khamis, M., Ngo, H. Q., Rudra, A.: FAQ: Questions asked frequently. In: Proceedings 35th ACM Symposium on Principles of Databases. pp. 13–28. ACM (2016)
12. Abo Khamis, M., Ngo, H. Q., Rudra, A.: Juggling functions inside a database. SIGMOD Rec. **46**(1), 6–13 (2017)
13. Maddux, R.: The origin of relation algebras in the development and axiomatization of the calculus of relations. Studia Logica **50**(3/4), 421–455 (1991)
14. Marx, M., Venema, Y.: Multi-Dimensional Modal Logic. Springer (1997)
15. Pratt, V.: Origins of the calculus of binary relations. In: Proceedings 7th Annual IEEE Symposium on Logic in Computer Science. pp. 248–254 (1992)
16. Tarski, A.: On the calculus of relations. J. Symb. Log. **6**, 73–89 (1941)
17. Tarski, A., Givant, S.: A Formalization of Set Theory Without Variables, AMS Colloquium Publications, vol. 41. American Mathematical Society (1987)
18. Yan, Z., Tannen, V., Ives, Z.: Fine-grained provenance for linear algebra operators. In: Proceedings 8th USENIX Workshop on the Theory and Practice of Provenance (2016)