On the Expressiveness of LARA: A Unified Language for Linear and Relational Algebra

Pablo Barceló
IMC, Pontificia Universidad Católica de Chile, Santiago, Chile
IMFD, Santiago, Chile
pbarcelo@ing.puc.cl

Nelson Higuera
Department of Computer Science, University of Chile, Santiago, Chile
IMFD, Santiago, Chile
nhiguera@dcc.uchile.cl

Jorge Pérez
Department of Computer Science, University of Chile, Santiago, Chile
IMFD, Santiago, Chile
jperez@dcc.uchile.cl

Bernardo Subercaseaux
Department of Computer Science, University of Chile, Santiago, Chile
IMFD, Santiago, Chile
bsubercaseaux@dcc.uchile.cl

Abstract

We study the expressive power of the Lara language – a recently proposed unified model for expressing relational and linear algebra operations – both in terms of traditional database query languages and some analytic tasks often performed in machine learning pipelines. We start by showing Lara to be expressive complete with respect to first-order logic with aggregation. Since Lara is parameterized by a set of user-defined functions which allow to transform values in tables, the exact expressive power of the language depends on how these functions are defined. We distinguish two main cases depending on the level of genericity queries are enforced to satisfy. Under strong genericity assumptions the language cannot express matrix convolution, a very important operation in current machine learning operations. This language is also local, and thus cannot express operations such as matrix inverse that exhibit a recursive behavior. For expressing convolution, one can relax the genericity requirement by adding an underlying linear order on the domain. This, however, destroys locality and turns the expressive power of the language much more difficult to understand. In particular, although under complexity assumptions the resulting language can still not express matrix inverse, a proof of this fact without such assumptions seems challenging to obtain.

2012 ACM Subject Classification Information systems → Query languages

Keywords and phrases languages for linear and relational algebra, expressive power, first order logic with aggregation, matrix convolution, matrix inverse, query genericity, locality of queries, safety

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.6

Related Version `https://arxiv.org/abs/1909.11693v1`

Funding This work is funded by the Millennium Institute for Foundational Research on Data (IMFD Chile). Barceló and Pérez are also funded by Fondecyt grant 1200967.

1 Introduction

Many of the actual analytics systems require both relational algebra and statistical functionalities for manipulating the data. In fact, while tools based on relational algebra are often used for preparing and structuring the data, the ones based on statistics and machine learning (ML) are applied to quantitatively reason about such data. Based on the "impedance mismatch" that this dichotomy creates [12], the database theory community has highlighted the need of developing a standard data model and query language for such applications, meaning an extension of relational algebra with linear algebra operators that is able to express the most common ML tasks [4]. Noticeably, the ML community has also recently manifested the need for what – at least from a database perspective – can be seen as a high-level language that manipulates tensors. Indeed, despite their wide adoption, there has been a recent interest in redesigning the way in which tensors are used in deep learning code [8, 15, 16], due to some pitfalls of the current way in which tensors are abstracted.

Hutchinson et al. [10, 9] have recently proposed a data model and a query language that aims at becoming the "universal connector" that solves the aforementioned impedance. On the one hand, the data model proposed corresponds to the so-called *associative tables*, which generalize relational tables, tensors, arrays, and others. Associative tables are two-sorted, consisting of *keys* and *values* that such keys map to. The query language, on the other hand, is called LARA, and subsumes several known languages for the data models mentioned above. LARA is an algebraic language designed in a minimalistic way by only including three operators; namely, *join*, *union*, and *extension*. In rough terms, the first one corresponds to the traditional join from relational algebra, the second one to the operation of aggregation, and the third one to the extension defined by a function as in a *flatmap* operation. It has been shown that LARA subsumes all relational algebra operations and is capable of expressing several interesting linear algebra operations used in graph algorithms [9].

Based on the proposal of LARA as a unified language for relational and linear algebra, it is relevant to develop a deeper understanding of its expressive power, both in terms of the logical query languages traditionally studied in database theory and ML operations often performed in practical applications. We start with the former and show that LARA is expressive complete with respect to *first-order logic with aggregation* ($\mathrm{FO}_{\mathsf{Agg}}$), a language that has been studied as a way to abstract the expressive power of SQL without recursion; cf., [13, 14]. (To be more precise, LARA is expressive complete with respect to a suitable syntactic fragment of $\mathrm{FO}_{\mathsf{Agg}}$ that ensures that formulas are *safe* and get properly evaluated over associative tables). This result can be seen as a sanity check for LARA. In fact, this language is specifically tailored to handle aggregation in conjunction with relational algebra operations, and a classical result in database theory establishes that the latter is expressive complete with respect to first-order logic (FO). We observe that while LARA consists of *positive* algebraic operators only, set difference can be encoded in the language by a combination of aggregate operators and extension functions. Our expressive completeness result is parameterized by the class of functions allowed in the extension operator. For each such a class Ω we allow $\mathrm{FO}_{\mathsf{Agg}}$ to contain all built-in predicates that encode the functions in Ω.

To understand which ML operators LARA can express, one then needs to bound the class Ω of extension functions allowed in the language. We start with a tame class that can still express several relevant functions. These are the FO-expressible functions that allow to compute arbitrary numerical predicates on values, but can only compare keys with respect to (in)equality. This restriction makes the logic quite amenable for theoretical exploration. In fact, it is easy to show that the resulting "tame version" of LARA satisfies a strong *genericity* criterion (in terms of key-permutations) and is also *local*, in the sense that queries in the language can only see up to a fixed-radius neighborhood from its free variables; cf., [14]. The first property implies that this tame version of LARA cannot express non-generic operations, such as matrix *convolution*, and the second one that it cannot express inherently recursive queries, such as matrix *inverse*. Both operations are very relevant for ML applications; e.g., matrix convolution is routinely applied in dimension-reduction tasks, while matrix inverse is used for learning the matrix of coefficient values in linear regression.

We then look more carefully at the case of matrix convolution, and show that this query can be expressed if we relax the genericity properties of the language by assuming the presence of a linear order on the domain of keys. (This relaxation implies that queries expressible in the resulting version of LARA are no longer invariant with respect to key-permutations). This language, however, is much harder to understand in terms of its expressive power. In particular, it can express non-local queries, and hence we cannot apply locality techniques to show that the matrix inversion query is not expressible in it. To prove this result, then, one would have to apply techniques based on the *Ehrenfeucht-Fraïssé* games that characterize the expressive power of the logic. Showing results based on such games in the presence of a linear order, however, is often combinatorially difficult, and currently we do not know whether this is possible. In turn, it is possible to obtain that matrix inversion is not expressible in a natural restriction of our language under complexity-theoretic assumptions. This is because the data complexity of queries expressible in such a restricted language is LOGSPACE, while matrix inversion is complete for a class that is believed to be a proper extension of the latter.

The main objective of our paper is connecting the study of the expressive power of tensor-based query languages, in general, and of LARA, in particular, with traditional database theory concepts and the arsenal of techniques that have been developed in this area to study the expressiveness of query languages. We also aim at identifying potential lines for future research that appear in connection with this problem. Our work is close in spirit to the recent study of MATLANG [1, 6], a matrix-manipulation language based on elementary linear algebra operations. It is shown that this language is contained in the three-variable fragment of relational algebra with summation and, thus, it is local. This implies that the core of MATLANG cannot check for the presence of a four-clique in a graph (represented as a Boolean matrix), as this query requires at least four variables to be expressed, and neither can it express the non-local matrix inversion query. It can be shown that MATLANG is strictly contained in the tame version of LARA that is mentioned above, and thus some of our results can be seen as generalizations of the ones for MATLANG.

Organization of the paper. Basics of LARA and $\mathrm{FO}_{\mathsf{Agg}}$ are presented in Sections 2 and 3, respectively. The expressive completeness of LARA in terms of $\mathrm{FO}_{\mathsf{Agg}}$ is shown in Section 4. The tame version of LARA and some inexpressibility results relating to it are given in Section 5, while in Section 6 we present a version of LARA that can express convolution and some discussion about its expressive power. We finalize in Section 7 with concluding remarks and future work. Due to space constraints some of our proofs are in the appendix, or simply reserved to a final version of the paper.

2 The LARA Language

For integers $m \leq n$, we write $[m,n]$ for $\{m,\dots,n\}$ and $[n]$ for $\{1,\dots,n\}$. If $\bar{v} = (v_1,\dots,v_n)$ is a tuple of elements, we write $\bar{v}[i]$ for v_i. We denote multisets as $\{\!\{a,b,\dots\}\!\}$.

Data model

A *relational schema* is a finite collection σ of *two-sorted* relation symbols. The first sort consists of *key-attributes* and the second one of *value-attributes*. Each relation symbol $R \in \sigma$ is then associated with a pair $(\bar{K},\bar{V})$, where $\bar{K}$ and $\bar{V}$ are (possibly empty) tuples of different key- and value- attributes, respectively. We write $R[\bar{K},\bar{V}]$ to denote that $(\bar{K},\bar{V})$ is the *sort* of R. We do not distinguish between $\bar{K}$, resp., $\bar{V}$, and the set of attributes mentioned in it.

There are two countably infinite sets of objects over which databases are populated: A domain of *keys*, which interpret key-attributes and is denoted Keys, and a domain of *values*, which interpret value-attributes and is denoted Values. A *tuple of sort* $(\bar{K}, \bar{V})$ is a function $t : \bar{K} \cup \bar{V} \to \mathsf{Keys} \cup \mathsf{Values}$ such that $t(A) \in \mathsf{Keys}$ if $A \in \bar{K}$ and $t(A) \in \mathsf{Values}$ if $A \in \bar{V}$. A *database* D over schema σ is a mapping that assigns with each relation symbol $R[\bar{K}, \bar{V}] \in \sigma$ a finite set R^D of tuples of sort $(\bar{K}, \bar{V})$. We often see D as a set of *facts*, i.e., as the set of expressions $R(t)$ such that $t \in R^D$. For ease of presentation, we write $R(\bar{k}, \bar{v}) \in D$ if $R(t) \in D$ for some tuple t with $t(\bar{K}) = \bar{k}$ and $t(\bar{V}) = \bar{v}$ (where $\bar{k} \in \mathsf{Keys}^{|\bar{K}|}$ and $\bar{v} \in \mathsf{Values}^{|\bar{V}|}$).

For a database D to be a Lara *database* we need D to satisfy an extra restriction: Key attributes define a key constraint over the corresponding relation symbols. That is,

$$R(\bar{k}, \bar{v}), R(\bar{k}, \bar{v}') \in D \quad \Longrightarrow \quad \bar{v} = \bar{v}',$$

for each $R[\bar{K}, \bar{V}] \in \sigma$, $\bar{k} \in \mathsf{Keys}^{|\bar{K}|}$, and $\bar{v}, \bar{v}' \in \mathsf{Values}^{|\bar{V}|}$. Relations of the form R^D are called *associative tables* [10]. Yet, we abuse terminology and call associative table any set A of tuples of the same sort $(\bar{K}, \bar{V})$ such that $\bar{v} = \bar{v}'$ for each $(\bar{k}, \bar{v}), (\bar{k}, \bar{v}') \in A$. In such a case, A is of sort $(\bar{K}, \bar{V})$. Notice that for a tuple $(\bar{k}, \bar{v})$ in A, we can safely denote $\bar{v} = A(\bar{k})$.

Syntax

An *aggregate operator* over domain U is a family $\oplus = \{\oplus_0, \oplus_1, \dots, \oplus_\omega\}$ of partial functions, where each $\oplus_k$ takes a multiset of k elements from U and returns a single element in U. If u is a collection of k elements in U, we write $\oplus(u)$ for $\oplus_k(u)$. This notion generalizes most aggregate operators used in practical query languages; e.g., SUM, AVG, MIN, MAX, and COUNT. For simplicity, we also see binary operations $\otimes$ on U as aggregate operators $\oplus = \{\oplus_0, \oplus_1, \dots, \oplus_\omega\}$ such that $\oplus_2 = \otimes$ and $\oplus_i$ has an empty domain for each $i \neq 2$.

The syntax of Lara is parameterized by a set of *extension functions*. This is a collection Ω of user-defined functions f that map each tuple t of sort $(\bar{K}, \bar{V})$ to a finite associative table of sort $(\bar{K}', \bar{V}')$, for $\bar{K} \cap \bar{K}' = \emptyset$ and $\bar{V} \cap \bar{V}' = \emptyset$. We say that f is of sort $(\bar{K}, \bar{V}) \mapsto (\bar{K}', \bar{V}')$. As an example, an extension function might take a tuple $t = (k, v_1, v_2)$ of sort (K, V_1, V_2), for $v_1, v_2 \in \mathbb{Q}$, and map it to a table of sort (K', V') that contains a single tuple (k, v), where v is the average between v_1 and v_2.

We inductively define the set of expressions in Lara(Ω) over schema σ as follows.

- *Empty associative table.* $\emptyset$ is an expression of sort $(\emptyset, \emptyset)$.
- *Atomic expressions.* If $R[\bar{K}, \bar{V}]$ is in σ, then R is an expression of sort $(\bar{K}, \bar{V})$.
- *Join.* If e_1 and e_2 are expressions of sort $(\bar{K}_1, \bar{V}_1)$ and $(\bar{K}_2, \bar{V}_2)$, respectively, and $\otimes$ is a binary operator over Values, then $e_1 \bowtie_\otimes e_2$ is an expression of sort $(\bar{K}_1 \cup \bar{K}_2, \bar{V}_1 \cup \bar{V}_2)$.
- *Union.* If e_1, e_2 are expressions of sort $(\bar{K}_1, \bar{V}_1)$ and $(\bar{K}_2, \bar{V}_2)$, respectively, and $\oplus$ is an aggregate operator over Values, then $e_1 \,⋈\!\!\!\!\!\! \times_\oplus\, e_2$ is an expression of sort $(\bar{K}_1 \cap \bar{K}_2, \bar{V}_1 \cup \bar{V}_2)$.
- *Extend.* For e an expression of sort $(\bar{K}, \bar{V})$ and f a function in Ω of sort $(\bar{K}, \bar{V}) \mapsto (\bar{K}', \bar{V}')$, it is the case that $\mathsf{Ext}_f\, e$ is an expression of sort $(\bar{K} \cup \bar{K}', \bar{V}')$.

We write $e[\bar{K}, \bar{V}]$ to denote that expression e is of sort $(\bar{K}, \bar{V})$.

Semantics

We assume that for every aggregate operator $\oplus$ over domain Values there is a *neutral value* $0_\oplus \in \mathsf{Values}$. Formally, $\oplus(\mathcal{V}) = \oplus(\mathcal{V}')$, for every multiset $\mathcal{V}$ of elements in Values and every extension $\mathcal{V}'$ of $\mathcal{V}$ with an arbitrary number of occurrences of $0_\oplus$. An important notion is *padding*. Let $\bar{V}_1$ and $\bar{V}_2$ be tuples of value-attributes, and $\bar{v}$ a tuple over Values of sort $\bar{V}_1$.

$\mathbf{i}$	$\mathbf{j}$	$\mathbf{v}_1$	$\mathbf{v}_2$
0	0	1	5
0	1	2	6
1	0	3	7
1	1	4	8

$\mathbf{j}$	$\mathbf{k}$	$\mathbf{v}_2$	$\mathbf{v}_3$
0	0	1	1
0	1	1	2
1	0	1	1
1	1	2	1

Figure 1 Associative tables A and B used for defining the semantics of Lara.

Then $\text{pad}_{\oplus}^{\bar{V}_2}(\bar{v})$ is a new tuple $\bar{v}'$ over Values of sort $\bar{V}_1 \cup \bar{V}_2$ such that for each $V \in \bar{V}_1 \cup \bar{V}_2$ we have that $v'(V) = v(V)$, if $V \in \bar{V}_1$, and $v'(V) = 0_{\oplus}$, otherwise.

Consider tuples $\bar{k}_1$ and $\bar{k}_2$ over key-attributes $\bar{K}_1$ and $\bar{K}_2$, respectively. We say that $\bar{k}_1$ and $\bar{k}_2$ are *compatible*, if $\bar{k}_1(K) = \bar{k}_2(K)$ for every $K \in \bar{K}_1 \cap \bar{K}_2$. If $\bar{k}_1$ and $\bar{k}_2$ are compatible, one can define the extended tuple $\bar{k}_1 \cup \bar{k}_2$ over key-attributes $\bar{K}_1 \cup \bar{K}_2$. Also, given a tuple $\bar{k}$ of sort $\bar{K}$, and a set $\bar{K}' \subseteq \bar{K}$, the restriction $\bar{k}{\downarrow_{\bar{K}'}}$ of $\bar{k}$ to attributes $\bar{K}'$ is the only tuple of sort $\bar{K}'$ that is compatible with $\bar{k}$. Finally, given a multiset T of tuples $(\bar{k}, \bar{u})$ of the same sort $(\bar{K}, \bar{V})$ we define $\mathsf{Solve}_{\oplus}(T)$ as

$$\begin{aligned}\mathsf{Solve}_{\oplus}(T) \;:=\; \{(\bar{k}, \bar{v}) \mid{} & \text{there exists } \bar{u} \text{ such that } (\bar{k}, \bar{u}) \in T \text{ and} \\ & \bar{v}[i] = \bigoplus_{\bar{v}' \,:\, (\bar{k}, \bar{v}') \in T} \bar{v}'[i], \text{ for each } i \in [|\bar{V}|]\}.\end{aligned}$$

That is, $\mathsf{Solve}_{\oplus}(T)$ turns the multiset T into an associative table T' by first grouping together tuples that have the same value over $\bar{K}$, and the solving key-conflicts by aggregating with respect to $\oplus$ (coordinate-wise).

The evaluation of a Lara(Ω) expression e over schema σ on a Lara database D, denoted e^D, is inductively defined as follows. Since the definitions are not so easy to grasp, we use the associative tables A and B in Figure 1 to construct examples. Here, $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ are key-attributes, while $\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{v}_3$ are value-attributes.

- *Empty associative table.* if $e = \emptyset$ then $e^D := \emptyset$.
- *Atomic expressions.* If $e = R[\bar{K}, \bar{V}]$, for $R \in \sigma$, then $e^D := R^D$.
- *Join.* If $e[\bar{K}_1 \cup \bar{K}_2, \bar{V}_1 \cup \bar{V}_2] = e_1[\bar{K}_1, \bar{V}_1] \bowtie_{\otimes} e_2[\bar{K}_2, \bar{V}_2]$, then

  $$\begin{aligned}e^D \;:=\; \{(\bar{k}_1 \cup \bar{k}_2, \bar{v}_1 \otimes \bar{v}_2) \mid{} & \bar{k}_1 \text{ and } \bar{k}_2 \text{ are compatible tuples such that} \\ & \bar{v}_1 = \text{pad}_{\otimes}^{\bar{V}_2}(e_1^D(\bar{k}_1)) \text{ and } \bar{v}_2 = \text{pad}_{\otimes}^{\bar{V}_1}(e_2^D(\bar{k}_2))\}.\end{aligned}$$

  Here, $\bar{v}_1 \otimes \bar{v}_2$ is a shortening for $(v_1^1 \otimes v_2^1, \ldots, v_n^1 \otimes v_n^2)$ assuming that $\bar{v}_1 = (v_1^1, \ldots, v_n^1)$ and $\bar{v}_2 = (v_2^1, \ldots, v_2^n)$. For example, the result of $A \bowtie_{\times} B$ is shown in Figure 2, for $\times$ being the usual product on $\mathbb{N}$ and $0_{\times} = 1$.
- *Union.* If $e[\bar{K}_1 \cap \bar{K}_2, \bar{V}_1 \cup \bar{V}_2] = e_1[\bar{K}_1, \bar{V}_1] \mathbin{\rotatebox[origin=c]{90}{$\bowtie$}}_{\oplus} e_2[\bar{K}_2, \bar{V}_2]$, then

  $$\begin{aligned}e^D \;:= \mathsf{Solve}_{\oplus}\{\!\{(\bar{k}, \bar{v}) \mid{} & \bar{k} = \bar{k}_1{\downarrow_{\bar{K}_1 \cap \bar{K}_2}} \text{ and } \bar{v} = \text{pad}_{\oplus}^{\bar{V}_2}(e_1^D(\bar{k}_1)) \text{ for some } \bar{k}_1 \in e_1^D, \\ & \text{or } \bar{k} = \bar{k}_2{\downarrow_{\bar{K}_1 \cap \bar{K}_2}} \text{ and } \bar{v} = \text{pad}_{\oplus}^{\bar{V}_1}(e_2^D(\bar{k}_2)) \text{ for some } \bar{k}_2 \in e_2^D\}\!\}.\end{aligned}$$

  In more intuitive terms, e^D is defined by first projecting over $\bar{K}_1 \cap \bar{K}_2$ any tuple in e_1^D, resp., in e_2^D. As the resulting set of tuples might no longer be an associative table (because there might be many tuples with the same keys), we have to solve the conflicts by applying the given aggregate operator $\oplus$. This is what $\mathsf{Solve}_{\oplus}$ does.
  For example, the result of $A \mathbin{\rotatebox[origin=c]{90}{$\bowtie$}}_{+} B$ is shown in Figure 2, for $+$ being the addition on $\mathbb{N}$.

(a) Table $A \bowtie_\times B$

i	**j**	**k**	**v**$_1$	**v**$_2$	**v**$_3$
0	0	0	1	5	1
0	0	1	1	5	2
0	1	0	2	6	1
0	1	1	2	12	1
1	0	0	3	7	1
1	0	1	3	7	2
1	1	0	4	8	1
1	1	1	4	16	1

(b) Table $A \,\Sigma_+\, B$

j	**v**$_1$	**v**$_2$	**v**$_3$
0	4	14	3
1	8	17	2

(c) Table $\mathsf{Ext}_g\, A$

i	**j**	**z**
0	0	1
0	1	1

Figure 2 The tables $A \bowtie_\times B$, $A \,\Sigma_+\, B$,and $\mathsf{Ext}_f\, A$.

- *Extend.* If $e[\bar{K} \cup \bar{K}', \bar{V}'] = \mathsf{Ext}_f\, e_1[\bar{K}, \bar{V}]$ and f is of sort $(\bar{K}, \bar{V}) \mapsto (\bar{K}', \bar{V}')$, then

  $$e^D := \{(\bar{k} \cup \bar{k}', \bar{v}') \mid (\bar{k}, \bar{v}) \in e_1^D, \text{ and } (\bar{k}', \bar{v}') \in f(\bar{k}, \bar{v})\}.$$

  Notice that in this case $\bar{k} \cup \bar{k}'$ always exists as $\bar{K} \cap \bar{K}' = \emptyset$.

  As an example, Figure 2 shows the results of $\mathsf{Ext}_g\, A$, where g is a function that does the following: If the key corresponding to attribute **i** is 0, then the tuple is associated with the associative table of sort $(\emptyset, \mathbf{z})$ containing only the tuple $(\emptyset, 1)$. Otherwise, the tuple is associated with the empty associative table.

Several useful operators, as described below, can be derived from the previous ones.

- *Map operation.* An important particular case of Ext_f occurs when f is of sort $(\bar{K}, \bar{V}) \mapsto (\emptyset, \bar{V}')$, i.e., when f does not extend the keys in the associative table but only modifies the values. Following [10], we write this operation as Map_f.
- *Aggregation.* This corresponds to an aggregation over some of the key-attributes of an associative table. Consider a LARA expression $e_1[\bar{K}_1, \bar{V}_1]$, an aggregate operator $\oplus$ over Values, and a $\bar{K} \subseteq \bar{K}_1$, then $e = \Sigma_\oplus^{\bar{K}}\, e_1$ is an expression of sort $(\bar{K}, \bar{V}_1)$ such that $e^D := \mathsf{Solve}_\oplus\{\{(\bar{k}, \bar{v}) \mid \bar{k} = \bar{k}_1{\downarrow_{\bar{K}}} \text{ and } \bar{v} = e_1^D(\bar{k}_1)\}\}$. We note that $\Sigma_\oplus^{\bar{K}}\, e_1$ is equivalent to the expression $e_1 \,\Sigma_\oplus\, \mathsf{Ext}_f(\emptyset)$, where f is the function that associates an empty table of sort $(\bar{K}, \emptyset)$ with every possible tuple.
- *Reduction.* The reduction operator, denoted by $\bar{\Sigma}$, is just a syntactic variation of aggregation defined as $\bar{\Sigma}_\oplus^{\bar{L}}\, e_1 \equiv \Sigma_\oplus^{\bar{K}\setminus\bar{L}}\, e_1$.

Next we provide an example that applies several of these operators.

▶ **Example 1.** Consider the schema Seqs$[(\mathit{time}, \mathit{batch}, \mathit{features}), (\mathit{val})]$, which represents a typical tensor obtained as the output of a recurrent neural network that processes input sequences. The structure stores a set of *features* obtained when processing input symbols from a sequence, one symbol at a *time*. For efficiency the network can simultaneously process a *batch* of examples and provide a single tensor as output.

Assume that, in order to make a prediction one wants to first obtain, for every example, the maximum value of every feature over the time steps, and then apply a *softmax* function.

One can specify all this process in LARA as follows.

$$
\begin{aligned}
\text{Max} &= \bar{\rotatebox[origin=c]{90}{$\bowtie$}}^{(\mathit{time})}_{\max(\cdot)}\ \text{Seqs} && (1)\\
\text{Exp} &= \mathsf{Map}_{\exp(\cdot)}\text{Max} && (2)\\
\text{SumExp} &= \bar{\rotatebox[origin=c]{90}{$\bowtie$}}^{(\mathit{features})}_{\text{sum}(\cdot)}\ \text{Exp} && (3)\\
\text{Softmax} &= \text{Exp} \bowtie_{\div} \text{SumExp} && (4)
\end{aligned}
$$

Expression (1) performs an aggregation over the *time* attribute to obtain the new tensor $\mathsf{Max}[(\mathit{batch}, \mathit{features}), (\mathit{val})]$ such that $\mathsf{Max}(b, f) = \max_{u=\mathsf{Seqs}(t,b,f)} u$. That is, Max stores the maximum value over all time steps (for every feature of every example). Expression (2) applies a point-wise exponential function to obtain the tensor $\mathsf{Exp}[(\mathit{batch}, \mathit{features}), (\mathit{val})]$ such that $\mathsf{Exp}(b, f) = \exp(\mathsf{Max}(b, f))$. In expression (3) we apply another aggregation to compute the sum of the exponentials of all the (maximum) features. Thus we obtain the tensor $\mathsf{SumExp}[(\mathit{batch}), (\mathit{val})]$ such that

$$\mathsf{SumExp}(b) = \sum_f \mathsf{Exp}(b, f) = \sum_f \exp(\mathsf{Max}(b, f)).$$

Finally, expression (4) applies point-wise division over the tensors $\mathsf{Exp}[(\mathit{batch}, \mathit{features}), (\mathit{val})]$ and $\mathsf{SumExp}[(\mathit{batch}), (\mathit{val})]$. This defines a tensor $\mathsf{Softmax}[(\mathit{batch}, \mathit{features}), (\mathit{val})]$ such that

$$\mathsf{Softmax}(b, f) = \frac{\mathsf{Exp}(b, f)}{\mathsf{SumExp}(b)} = \frac{\exp(\mathsf{Max}(b, f))}{\sum_{f'} \exp(\mathsf{Max}(b, f'))}.$$

Thus, we effectively compute the *softmax* of the vector of maximum features over time for every example in the batch. ◀

It is easy to see that for each LARA expression e and LARA database D, the result $e(D)$ is always an associative table. Moreover, although the elements in the evaluation $e(D)$ of an expression e over D are not necessarily in D (due to the applications of the operator $\mathsf{Solve}_{\oplus}$ and the extension functions in Ω), all LARA expressions are *safe*, i.e., $|e^D|$ is finite.

▶ **Proposition 2.** *Let e be a LARA(Ω) expression. Then e^D is a finite associative table, for every LARA database D.*

3 First-order Logic with Aggregation

We consider a two-sorted version of FO with aggregation. We thus assume the existence of two disjoint and countably infinite sets of *key-variables* and *value-variables*. The former are denoted $x, y, z, \ldots$ and the latter $i, j, k, \ldots$. In order to cope with the demands of the extension functions used by LARA (as explained later), we allow the language to be parameterized by a collection Ψ of user-defined relations R of some sort $(\bar{K}, \bar{V})$. For each $R \in \Psi$ we blur the distinction between the symbol R and its interpretation over $\mathsf{Keys}^{|\bar{K}|} \times \mathsf{Values}^{|\bar{V}|}$.

Syntax and semantics

The language contains terms of two sorts.

- *Key-terms*: Composed exclusively by the key-variables $x, y, z \ldots$.
- *Value-terms*: Composed by the constants of the form $0_{\oplus}$, for each aggregate operator $\oplus$, the value-variables $i, j, \ldots$, and the *aggregation terms* defined next. Let $\tau(\bar{x}, \bar{y}, \bar{i}, \bar{j})$ be a value-term mentioning only key-variables among those in $(\bar{x}, \bar{y})$ and value-variables

among those in $(\bar{i}, \bar{j})$, and $\phi(\bar{x}, \bar{y}, \bar{i}, \bar{j})$ a formula whose *free* key- and value-variables are those in $(\bar{x}, \bar{y})$ and $(\bar{i}, \bar{j})$, respectively (i.e., the variables that do not appear under the scope of a quantifier). Then for each aggregate operator $\oplus$ we have that

$$\tau'(\bar{x}, \bar{i}) \;:=\; \mathsf{Agg}_{\oplus}\bar{y}, \bar{j}\,\big(\tau(\bar{x}, \bar{y}, \bar{i}, \bar{j}), \phi(\bar{x}, \bar{y}, \bar{i}, \bar{j})\big) \qquad (5)$$

is a value-term whose free variables are those in $\bar{x}$ and $\bar{i}$.

Let Ψ be a set of relations R as defined above. The set of formulas in the language $\mathrm{FO}_{\mathsf{Agg}}(\Psi)$ over schema σ is inductively defined as follows:

- Atoms $\perp$, $x = y$, and $\iota = \kappa$ are formulas, for x, y key-variables and ι, κ value-terms.
- If $R[\bar{K}, \bar{V}] \in \sigma \cup \Psi$, then $R(\bar{x}, \bar{\iota})$ is a formula, where $\bar{x}$ is a tuple of key-variables of the same arity as $\bar{K}$ and $\bar{\iota}$ is a tuple of value-terms of the same arity as $\bar{V}$.
- If ϕ, ψ are formulas, then $(\neg\phi)$, $(\phi \vee \psi)$, $(\phi \wedge \psi)$, $\exists x\phi$, and $\exists i\phi$ are formulas, where x and i are key- and value-variables, respectively.

We now define the semantics of $\mathrm{FO}_{\mathsf{Agg}}(\Psi)$. Let D be a LARA database and η an *assignment* that interprets each key-variable x as an element $\eta(x) \in \mathsf{Keys}$ and value-variable i as an element $\eta(i) \in \mathsf{Values}$. If $\tau(\bar{x}, \bar{i})$ is a value-term only mentioning variables in $(\bar{x}, \bar{i})$, we write $\tau^D(\eta(\bar{x}, \bar{i}))$ for the *interpretation* of τ over D when variables are interpreted according to η. Also, if $\phi(\bar{x}, \bar{i})$ is a formula of the logic whose free key- and value-variables are those in $(\bar{x}, \bar{i})$, we write $D \models \phi(\eta(\bar{x}, \bar{i}))$ if D *satisfies* ϕ when $\bar{x}, \bar{i}$ is interpreted according to η, and ϕ^D for the set of tuples $\eta(\bar{x}, \bar{i})$ such that $D \models \phi(\eta(\bar{x}, \bar{i}))$ for some assignment η.

The notion of satisfaction is inherited from the semantics of two-sorted FO. The notion of interpretation, on the other hand, requires explanation for the case of value-terms. Let η be an assignment as defined above. Constants $0_{\oplus}$ are interpreted as themselves and value-variables are interpreted over Values according to η. Consider now an aggregate term of the form (5). Let D be a LARA database and assume that $\eta(\bar{x}) = \bar{k}$, for $\bar{k} \in \mathsf{Keys}^{|\bar{x}|}$, and $\eta(\bar{i}) = \bar{v}$, for $\bar{v} \in \mathsf{Values}^{|\bar{i}|}$. Let $(\bar{k}'_1, \bar{v}'_1), (\bar{k}'_2, \bar{v}'_2), \dots,$ be an enumeration of all tuples $(\bar{k}', \bar{v}') \in \mathsf{Keys}^{|\bar{y}|} \times \mathsf{Values}^{|\bar{j}|}$ such that $D \models \phi(\bar{k}, \bar{k}', \bar{v}, \bar{v}')$, i.e. there is an assignment η' that coincides with η over all variables in $(\bar{x}, \bar{i})$ and satisfies $\eta'(\bar{y}, \bar{j}) = (\bar{k}', \bar{v}')$. Then

$$\tau'(\eta(\bar{x}, \bar{i})) \;=\; \tau'(\bar{k}, \bar{v}) \;:=\; \bigoplus \{\!\{\tau(\bar{k}, \bar{k}'_1, \bar{v}, \bar{v}'_1), \tau(\bar{k}, \bar{k}'_2, \bar{v}, \bar{v}'_2), \dots\}\!\} \in \mathsf{Values}.$$

4 Expressive Completeness of LARA with respect to $\mathrm{FO}_{\mathsf{Agg}}$

We prove that $\mathrm{LARA}(\Omega)$ has the same expressive power as a suitable restriction of $\mathrm{FO}_{\mathsf{Agg}}(\Psi_\Omega)$, where Ψ_Ω is a set that contains relations that represent the extension functions in Ω. In particular, for every extension function $f \in \Omega$ of sort $(\bar{K}, \bar{V}) \mapsto (\bar{K}', \bar{V}')$, there is a relation $R_f \subseteq \mathsf{Keys}^{|\bar{K}|+|\bar{K}'|} \times \mathsf{Values}^{|\bar{V}|+|\bar{V}'|}$ in Ψ_Ω such that for every $(\bar{k}, \bar{v}) \in \mathsf{Keys}^{|\bar{K}|} \times \mathsf{Values}^{|\bar{V}|}$:

$$f(\bar{k}, \bar{v}) \;=\; \{(\bar{k}', \bar{v}') \mid (\bar{k}, \bar{k}', \bar{v}, \bar{v}') \in R_f\}.$$

Since LARA is defined in a minimalistic way, we require some assumptions for our expressive completeness result to hold. First, we assume that $\mathsf{Keys} = \mathsf{Values}$, which allows us to interchangeably move from keys to values in the language (an operation that LARA routinely performs in several of its applications [10, 9]). Moreover, we assume that there are two reserved values, $\diamondsuit$ and $\heartsuit$, which are allowed to be used as constants in both $\mathrm{FO}_{\mathsf{Agg}}(\Psi_\Omega)$ and LARA, but do not appear in any LARA database. In particular, we allow $\mathrm{FO}_{\mathsf{Agg}}(\Psi_\Omega)$ to express formulas of the form $x = c$ and $i = c$, for x and i key- and value-variables, respectively, and $c \in \{\diamondsuit, \heartsuit\}$. (Notice that, by definition, $\diamondsuit$ and $\heartsuit$ are different to all neutral elements of the form $0_\oplus$, for $\oplus$ an aggregate operator). With these constants we can "mark" tuples in some specific cases, and thus solve an important semantic mismatch between the two languages. In fact, LARA deals with multisets in their semantics while FO_{Agg} is based on sets only. This causes problems, e.g., when taking the union of two associative tables A and B both of which contain an occurrence of the same tuple $(\bar{k}, \bar{v})$. While LARA would treat both occurrences of $(\bar{k}, \bar{v})$ as different in $A \between B$, and hence would be forced to restore the "key-functionality" of $\bar{k}$ based on some aggregate operator, for FO_{Agg} the union of A and B contains only one occurrence of $(\bar{k}, \bar{v})$.

From LARA to FO_{Agg}

We show first that the expressive power of $\text{LARA}(\Omega)$ is bounded by that of $\text{FO}_{\mathsf{Agg}}(\Psi_\Omega)$.

▶ **Theorem 3.** *For every expression $e[\bar{K}, \bar{V}]$ of $\text{LARA}(\Omega)$ there is a formula $\phi_e(\bar{x}, \bar{i})$ of $\text{FO}_{\mathsf{Agg}}(\Psi_\Omega)$ such that $e^D = \phi_e^D$, for every LARA database D.*

Before proving the theorem, we present an example of how the join operation is translated, as this provides a good illustration of the main ideas behind the proof. Assume that we are given expressions $e_1[K_1, K_2, K_3, V_1, V_2]$ and $e_2[K_3, K_2, V_2, V_3]$ of $\text{LARA}(\Omega)$. Inductively, there are formulas $\phi_{e_1}(x_1, x_2, x_3, i_1, i_2)$ and $\phi_{e_2}(x_3', x_2', i_2', i_3)$ of $\text{FO}_{\mathsf{Agg}}(\Psi_\Omega)$ such that $e_1^D = \phi_{e_1}^D$ and $e_2^D = \phi_{e_2}^D$, for every LARA database D. We want to be able to express $e = e_1 \bowtie_\otimes e_2$ in $\text{FO}_{\mathsf{Agg}}(\Psi_\Omega)$. Let us define a formula $\alpha(x, y, z, i, j, k, f)$ as

$$\exists i', j', k' \Big(\phi_{e_1}(x, y, z, i', j') \wedge \phi_{e_2}(y, z, j', k') \wedge \\ \big((i = i' \wedge j = j' \wedge k = 0_\otimes \wedge f = \diamondsuit) \vee (i = 0_\otimes \wedge j = j' \wedge k = k' \wedge f = \heartsuit)\big) \Big).$$

Notice that when evaluating α over a LARA database D we obtain the set of tuples $(k_1, k_2, k_3, v_1, v_2, v_3, f)$ such that there exist tuples of the form $(k_1, k_2, k_3, \cdot, \cdot) \in e_1^D$ and $(k_2, k_3, \cdot, \cdot) \in e_2^D$, and either one of the following holds:

- $e_1^D(k_1, k_2, k_3) = (v_1, v_2)$; $(v_1, v_2, v_3) = \text{pad}_\otimes^{(V_2, V_3)}(v_1, v_2)$; and $f = \diamondsuit$; or
- $e_2^D(k_2, k_3) = (v_2, v_3)$; $(v_1, v_2, v_3) = \text{pad}_\otimes^{(V_1, V_2)}(v_2, v_3)$; and $f = \heartsuit$.

The reason why we want to distinguish tuples from e_1^D or e_2^D with a $\diamondsuit$ or a $\heartsuit$ in the position of variable f, respectively, it is because it could be the case that $\text{pad}_\otimes^{(V_2, V_3)}(v_1, v_2) = \text{pad}_\otimes^{(V_1, V_2)}(v_2, v_3)$. The semantics of LARA, which is based on aggregation over multisets of tuples, forces us to treat them as two different tuples. The way we do this in FO_{Agg} is by distinguishing them with the extra flag f.

We finally define $\phi_e(x, y, z, i, j, k)$ in $\text{FO}_{\mathsf{Agg}}(\Psi_\Omega)$ as

$$\exists i', j', k', f' \Big(\alpha(x, y, z, i', j', k', f) \wedge i = \mathsf{Agg}_\otimes i', j', k', f' \left(i',\, \alpha(x, y, z, i', j', k', f')\right) \wedge \\ j = \mathsf{Agg}_\otimes i', j', k', f' \left(j',\, \alpha(x, y, z, i', j', k', f')\right) \wedge \\ k = \mathsf{Agg}_\otimes i', j', k', f' \left(k',\, \alpha(x, y, z, i', j', k', f')\right) \Big).$$

That is, the evaluation of ϕ_e on D outputs all tuples $(k_1, k_2, k_3, v_1, v_2, v_3)$ such that:

1. There are tuples $(k_1, k_2, k_3, w_1, w_2, 0_\otimes, \cdot, \diamondsuit)$ and $(k_1, k_2, k_3, 0_\otimes, w_2', w_3', \heartsuit)$ in α^D.
2. $v_1 = w_1 \otimes 0_\otimes = w_1$; $v_2 = w_2 \otimes w_2'$; and $v_3 = 0_\otimes \otimes w_3' = w_3'$.

Clearly, then, $\phi_e^D = e^D$ over every LARA database D.

Proof of Theorem 3. By induction on e.

- If $e = \emptyset$, then $\phi_e = \bot$.
- If $e = R[\bar{K}, \bar{V}]$, for $R \in \sigma$, then $\phi_e(\bar{x}, \bar{i}) = R(\bar{x}, \bar{i})$, where $\bar{x}$ and $\bar{i}$ are tuples of distinct key- and value-variables of the same arity as $\bar{K}$ and $\bar{V}$, respectively.
- Consider the expression $e[\bar{K}_1 \cup \bar{K}_2, \bar{V}_1 \cup \bar{V}_2] = e_1[\bar{K}_1, \bar{V}_1] \bowtie_{\otimes} e_2[\bar{K}_2, \bar{V}_2]$, and assume that $\phi_{e_1}(\bar{x}_1, \bar{i}_1)$ and $\phi_{e_2}(\bar{x}_2, \bar{i}_2)$ are the formulas obtained for $e_1[\bar{K}_1, \bar{V}_1]$ and $e_2[\bar{K}_2, \bar{V}_2]$, respectively, by induction hypothesis. Let us first define a formula $\alpha_e(\bar{x}_1, \bar{x}_2, \bar{j}, f)$ as

$$\exists \bar{i}_1, \bar{i}_2 \Big(\phi_{e_1}(\bar{x}_1, \bar{i}_1) \wedge \phi_{e_2}(\bar{x}_2, \bar{i}_2) \wedge \chi_{\bar{K}_1 \cap \bar{K}_2}(\bar{x}_1, \bar{x}_2) \wedge \\ \big((\bar{j} = \text{pad}_{\otimes}^{\bar{V}_2}(\bar{i}_1) \wedge f = \diamondsuit) \vee (\bar{j} = \text{pad}_{\otimes}^{\bar{V}_1}(\bar{i}_2) \wedge f = \heartsuit) \big) \Big),$$

  assuming that $\bar{x}_1$ and $\bar{x}_2$ share no variables; the same holds for $\bar{i}_1$ and $\bar{i}_2$; and the formula $\chi_{\bar{K}_1 \cap \bar{K}_2}(\bar{x}_1, \bar{x}_2)$ takes the conjunction of all atomic formulas of the form $y_1 = y_2$, where y_1 and y_2 are variables that appear in $\bar{x}_1$ and $\bar{x}_2$, respectively, in the position of some attribute $K \in \bar{K}_1 \cap \bar{K}_2$. Notice that $\bar{j} = \text{pad}_{\otimes}^{\bar{V}_2}(\bar{i}_1)$ is expressible in $\text{FO}_{\mathsf{Agg}}(\Psi_\Omega)$ as a conjunction of formulas of the form $j = k$, for each $j \in \bar{j}$, where k is the corresponding variable of $\bar{i}_1$ if j falls in the position of an attribute in $\bar{V}_1$, and it is is $0_{\otimes}$ otherwise. For instance, if $\bar{i} = (i_1, i_2)$ is of sort $\bar{V}_1 = (V_1, V_2)$ and $\bar{V}_2 = (V_2, V_3)$, then $\bar{j} = \text{pad}_{\otimes}^{\bar{V}_2}(\bar{i})$ is the formula $j_1 = i_1 \wedge j_2 = i_2 \wedge j_3 = 0_{\otimes}$. Analogously we can define $\bar{j} = \text{pad}_{\otimes}^{\bar{V}_1}(\bar{i}_2)$. As in the example given before this proof, we use the value of f to distinguish whether a tuple comes from e_1^D or e_2^D.
  It is easy to see that for every LARA database D we have that α_e^D computes all tuples of the form $(\bar{k}_1, \bar{k}_2, \bar{v}, w)$ such that $\bar{k}_1 \in (\exists \bar{i}_1 \phi_{e_1})^D$, $\bar{k}_2 \in (\exists \bar{i}_2 \phi_{e_2})^D$, and $\bar{k}_1$ and $\bar{k}_2$ are compatible tuples, and either one of the following statements hold:
  - $\bar{v}_1 = e_1^D(\bar{k}_1)$, $\bar{v} = \text{pad}_{\otimes}^{\bar{V}_2}(\bar{v}_1)$, and $w = \diamondsuit$; or
  - $\bar{v}_2 = e_2^D(\bar{k}_2)$, $\bar{v} = \text{pad}_{\otimes}^{\bar{V}_1}(\bar{v}_2)$, and $w = \heartsuit$.

  Notice then that for every $(\bar{k}_1, \bar{k}_2)$ that belongs to the evaluation of $\exists \bar{j}, f\, \alpha_e(\bar{x}_1, \bar{x}_2, \bar{j}, f)$ over D there are exactly two tuples of the form $(\bar{k}_1, \bar{k}_2, \bar{v}, w) \in \alpha_e^D$: the one which satisfies $\bar{v} = \text{pad}_{\otimes}^{\bar{V}_2}(e_1^D(\bar{k}_1))$ and $w = \diamondsuit$, and the one that satisfies $\bar{v} = \text{pad}_{\otimes}^{\bar{V}_1}(e_2^D(\bar{k}_2))$ and $w = \heartsuit$. It should be clear then that $\phi_e(\bar{x}_1, \bar{x}_2, \bar{i})$ can be expressed as:

$$\exists \bar{j}, f\, \alpha_e(\bar{x}_1, \bar{x}_2, \bar{j}, f) \wedge \bigwedge_{\ell \in [|\bar{j}|]} \bar{i}[\ell] = \mathsf{Agg}_{\otimes} \bar{j}, f\, (\bar{j}[\ell], \alpha_e(\bar{x}, \bar{j}, f)).$$

  Notice here that the aggregation is always performed on multisets with exactly two elements (by our previous observation). Clearly, the evaluation of ϕ_e on D, for D a LARA database, contains all tuples $(\bar{k}_1 \cup \bar{k}_2, \bar{v}_1 \otimes \bar{v}_2)$ such that $\bar{k}_1$ and $\bar{k}_2$ are compatible tuples in e_1^D and e_2^D, respectively, and it is the case that $\bar{v}_1 = \text{pad}_{\otimes}^{\bar{V}_2}(e_1^D(\bar{k}_1))$ and $\bar{v}_2 = \text{pad}_{\otimes}^{\bar{V}_1}(e_2^D(\bar{k}_2))$.
- Consider the expression $e[\bar{K}_1 \cap \bar{K}_2, \bar{V}_1 \cup \bar{V}_2] = e_1[\bar{K}_1, \bar{V}_1] \,⧖_{\oplus}\, e_2[\bar{K}_2, \bar{V}_2]$, and assume that $\phi_{e_1}(\bar{x}_1, \bar{i}_1)$ and $\phi_{e_2}(\bar{x}_2, \bar{i}_2)$ are the formulas obtained for $e_1[\bar{K}_1, \bar{V}_1]$ and $e_2[\bar{K}_2, \bar{V}_2]$, respectively, by induction hypothesis. We start by defining a formula $\alpha_{e_1}(\bar{x}, \bar{j}, f)$ as

$$\exists \bar{x}_1, \bar{i}_1 \left(\phi_{e_1}(\bar{x}_1, \bar{i}_1) \wedge \eta_1^{\bar{K}_1 \cap \bar{K}_2}(\bar{x}, \bar{x}_1) \wedge \bar{j} = \text{pad}_{\oplus}^{\bar{V}_2}(\bar{i}_1) \wedge f = \diamondsuit \right),$$

where $\eta_1^{\bar{K}_1\cap\bar{K}_2}(\bar{x},\bar{x}_1)$ states that $\bar{x}$ is the extension of $\bar{x}_1$ to represent a tuple in $\bar{K}_1\cup\bar{K}_2$. Formally, each variable in $\bar{x}$ that represents a position in $\bar{K}_1$ receives the same value than the variable in the corresponding position of $\bar{x}$, and each variable representing a position in $\bar{K}_2\setminus\bar{K}_1$ receives value $\diamondsuit$. As an example, if $\bar{K}_1=(K_1,K_2)$ and $\bar{K}_2=(K_1,K_3)$, then $\eta_1^{\bar{K}_1\cup\bar{K}_2}(\bar{x},\bar{x}_1)$ for $\bar{x}_1=(y_1,y_2)$ and $\bar{x}=(z_1,z_2,z_3)$ is $z_1=y_1\wedge z_2=y_2\wedge z_3=0$. Analogously, we define $\alpha_{e_2}(\bar{x},\bar{j},f)$ as

$$\exists\bar{x}_2,\bar{i}_2\left(\phi_{e_2}(\bar{x}_2,\bar{i}_2)\wedge\eta_2^{\bar{K}_1\cap\bar{K}_2}(\bar{x},\bar{x}_2)\wedge\bar{j}=\mathrm{pad}_{\oplus}^{\bar{V}_1}(\bar{i}_2)\wedge f=\heartsuit\right).$$

As before, we use distinguished constants $\diamondsuit$ and $\heartsuit$ as a way to distinguish tuples coming from e_1^D and e_2^D, respectively.
Let us now define $\alpha(\bar{x},\bar{j},f):=\alpha_{e_1}(\bar{x},\bar{j},f)\vee\alpha_{e_2}(\bar{x},\bar{j},f)$. It is not hard to see then that the evaluation of α_e on a Lara database D consists precisely of the tuples of the form $(\bar{k},\bar{v},w)$ such that one of the following statements hold:

- $\bar{k}_1=\bar{k}{\downarrow}_{\bar{K}_1}$ for some $\bar{k}_1\in(\exists\bar{i}_1\phi_{e_1})^D$; $\bar{k}$ takes value $\diamondsuit$ for those positions in $\bar{K}_2\setminus\bar{K}_1$; $\bar{v}=\mathrm{pad}_{\oplus}^{\bar{V}_2}(e_1^D(\bar{k}_1))$; and $w=\diamondsuit$; or
- $\bar{k}_2=\bar{k}{\downarrow}_{\bar{K}_2}$ for some $\bar{k}_2\in(\exists\bar{i}_2\phi_{e_2})^D$; $\bar{k}$ takes value $\heartsuit$ for those positions in $\bar{K}_1\setminus\bar{K}_2$; $\bar{v}=\mathrm{pad}_{\oplus}^{\bar{V}_1}(e_2^D(\bar{k}_1))$; and $w=\heartsuit$.

Let us write $\alpha(\bar{x},\bar{j},f)$ as $\alpha(\bar{x}',\bar{x}'',\bar{j},f)$ to denote that $\bar{x}'$ is the subtuple of $\bar{x}$ that corresponds to variables in $\bar{K}_1\cap\bar{K}_2$, while $\bar{x}''$ contains all other variables in $\bar{x}$. Notice that in the output of our desired formula ϕ_e we are only interested in the value that takes the tuple $\bar{x}'$. It should be clear then that $\phi_e(\bar{x},\bar{i})$ can be expressed as:

$$\exists\bar{x}',\bar{x}'',\bar{j},f\,\Big(\bar{x}=\bar{x}'\wedge\alpha(\bar{x}',\bar{x}'',\bar{j},f)\wedge\bigwedge_{\ell\in[|\bar{j}|]}\bar{i}[\ell]=\mathsf{Agg}_{\oplus}\bar{x}'',\bar{j},f\,(\bar{j}[\ell],\alpha(\bar{x}',\bar{x}'',\bar{j},f))\Big).$$

- Consider the expression $e[\bar{K}\cup\bar{K}',\bar{V}']=\mathsf{Ext}_f\,e_1[\bar{K},\bar{V}]$, where f is of sort $(\bar{K},\bar{V})\mapsto(\bar{K}',\bar{V}')$, and assume that $\phi_{e_1}(\bar{x}_1,\bar{i}_1)$ is the formula obtained for $e_1[\bar{K}_1,\bar{V}_1]$ by induction hypothesis. It is straightforward to see then that we can define

$$\phi_e(\bar{x}_1,\bar{x},\bar{i}):=\exists\bar{i}_1\left(\phi_{e_1}(\bar{x}_1,\bar{i}_1)\wedge R_f(\bar{x}_1,\bar{x},\bar{i}_1,\bar{i})\right).$$

This finishes the proof of the theorem. ◀

It is worth noticing that the translation from Lara to $\mathrm{FO}_{\mathsf{Agg}}$ given in the proof of Theorem 3 does not require the use of negation. In the next section we show that at least safe negation can be encoded in Lara by a suitable combination of aggregate operators and extension functions.

From $\mathrm{FO}_{\mathsf{Agg}}$ to LARA

We now prove that the other direction holds under suitable restrictions and assumptions on the language. First, we need to impose two restrictions on $\mathrm{FO}_{\mathsf{Agg}}$ formulas, which ensure that the semantics of the formulas considered matches that of Lara. In particular, we need to ensure that the evaluation of $\mathrm{FO}_{\mathsf{Agg}}$ formulas is safe and only outputs associative tables.

- *Safety.* Formulas of $\mathrm{FO}_{\mathsf{Agg}}(\Psi_\Omega)$ are not necessarily safe, i.e., their evaluation can have infinitely many tuples (think, e.g., of the formula $i=j$, for i,j value-variables, or $R_f(\bar{x},\bar{x}',\bar{i},\bar{i}')$, for $R_f\in\Psi_\Omega$). While safety issues relating to the expressive completeness of relational algebra with respect to first order logic are often resolved by relativizing

all operations to the *active* domain of databases (i.e., the set of elements mentioned in relations in databases), such a restriction only makes sense for keys in our context, but not for values. In fact, several useful formulas compute a new value for a variable based on some aggregation terms over precomputed data (see, e.g., the translations of the join and union operator of LARA into $\mathrm{FO}_{\mathsf{Agg}}$ provided in the proof of Theorem 3).
To overcome this issue we develop a suitable syntactic restriction of the logic that can only express safe queries. This is achieved by "guarding" the application of value-term equalities, relations encoding extension functions, and Boolean connectives as follows.

- We only allow equality of value-terms to appear in formulas of the form $\phi(\bar{x}, \bar{i}) \wedge j = \tau(\bar{x}, \bar{i})$, where j is a value-variable that does not necessarily appear in $\bar{i}$ and τ is an arbitrary value-term whose value only depends on $(\bar{x}, \bar{i})$. This formula computes the value of the aggregated term τ over the precomputed evaluation of ϕ, and then output it as the value of j. In the same vein, atomic formulas of the form $R(\bar{x}, \bar{\iota})$ must satisfy that every element in $\bar{\iota}$ is a value-variable.
- Relations $R_f \in \Psi_\Omega$ can only appear in formulas of the form $\phi(\bar{x}, \bar{i}) \wedge R_f(\bar{x}, \bar{x}', \bar{i}, \bar{i}')$, i.e., we only allow to compute the set $f(\bar{x}, \bar{i})$ for specific precomputed values of $(\bar{x}, \bar{i})$.
- Also, negation is only allowed in the restricted form $\phi(\bar{x}, \bar{i}) \wedge \neg\psi(\bar{x}, \bar{i})$ and disjunction in the form $\phi(\bar{x}, \bar{i}) \vee \psi(\bar{x}, \bar{i})$, i.e., when formulas have exactly the same free variables.

We denote the resulting language as $\mathrm{FO}^{\mathrm{safe}}_{\mathsf{Agg}}(\Psi_\Omega)$. These restrictions are meaningful, as the translation from $\mathrm{LARA}(\Omega)$ to $\mathrm{FO}_{\mathsf{Agg}}(\Omega_\Psi)$ given in the proof of Theorem 3 always builds a formula in $\mathrm{FO}^{\mathrm{safe}}_{\mathsf{Agg}}(\Psi_\Omega)$.

- *Key constraints.* We also need a restriction on the interpretation of $\mathrm{FO}^{\mathrm{safe}}_{\mathsf{Agg}}(\Psi_\Omega)$ formulas that ensures that the evaluation of any such a formula on a LARA database is an associative table. For doing this, we modify the syntax of $\mathrm{FO}^{\mathrm{safe}}_{\mathsf{Agg}}(\Psi_\Omega)$ formulas in such a way that every formula ϕ of $\mathrm{FO}^{\mathrm{safe}}_{\mathsf{Agg}}(\Psi_\Omega)$ now comes equipped with an aggregate operator $\oplus$ over Values. The operator $\oplus$ is used to "solve" the key violations introduced by the evaluation of ϕ. Thus, formulas in this section should be understood as pairs $(\phi, \oplus)$. The evaluation of $(\phi, \oplus)$ over a LARA database D, denoted $\phi^D_\oplus$, is $\mathsf{Solve}_\oplus(\phi^D)$. This definition is recursive; e.g., a formula in $\phi(\bar{x}, \bar{i})$ in $\mathrm{FO}^{\mathrm{safe}}_{\mathsf{Agg}}(\Psi_\Omega)$ which is of the form $\alpha(\bar{x}, \bar{i}) \vee \beta(\bar{x}, \bar{i})$ should now be specified as $(\phi, \oplus) = (\alpha, \oplus_\alpha) \vee (\beta, \oplus_\beta)$. The associative table $\phi^D_\oplus$ corresponds then to

$$\mathsf{Solve}_\oplus(\alpha^D_{\oplus_\alpha} \cup \beta^D_{\oplus_\beta}).$$

We also require some natural assumptions on the extension functions that LARA is allowed to use. In particular, we need these functions to be able to express traditional relational algebra operations that are not included in the core of LARA; namely, copying attributes, selecting rows based on (in)equality, and projecting over value-attributes (the projection over key-attributes, in turn, can be expressed with the union operator). Formally, we assume that Ω contains the following families of extension functions.

- $\mathsf{copy}_{\bar{K},\bar{K}'}$ and $\mathsf{copy}_{\bar{V},\bar{V}'}$, for $\bar{K}, \bar{K}'$ tuples of key-attributes of the same arity and $\bar{V}, \bar{V}'$ tuples of value-attributes of the same arity. Function $\mathsf{copy}_{\bar{K},\bar{K}'}$ takes as input a tuple $t = (\bar{k}, \bar{v})$ of sort $(\bar{K}_1, \bar{V})$, where $\bar{K} \subseteq \bar{K}_1$ and $\bar{K}' \cap \bar{K}_1 = \emptyset$, and produces a tuple $t' = (\bar{k}, \bar{k}', \bar{v})$ of sort $(\bar{K}_1, \bar{K}', \bar{V})$ such that $t'(\bar{K}') = t(\bar{K})$, i.e., $\mathsf{copy}_{\bar{K},\bar{K}'}$ copies the value of attributes $\bar{K}$ in the new attributes $\bar{K}'$. Analogously, we define the function $\mathsf{copy}_{\bar{V},\bar{V}'}$.
- $\mathsf{copy}_{\bar{V},\bar{K}}$, for $\bar{V}$ a tuple of value-attributes and $\bar{K}$ a tuple of key-attributes. It takes as input a tuple $t = (\bar{k}, \bar{v})$ of sort $(\bar{K}_1, \bar{V}_1)$, where $\bar{V} \subseteq \bar{V}_1$ and $\bar{K} \cap \bar{K}_1 = \emptyset$, and produces a tuple $t' = (\bar{k}, \bar{k}', \bar{v})$ of sort $(\bar{K}_1, \bar{K}, \bar{V})$ such that $t'(\bar{V}) = t'(\bar{K})$, i.e., this function copies the values in $\bar{V}$ as keys in $\bar{K}$. (Here it is important our assumption that $\mathsf{Keys} = \mathsf{Values}$). The reason why this is useful is because LARA does not allow to aggregate with respect to values (only with respect to keys), while $\mathrm{FO}_{\mathsf{Agg}}(\Psi_\Omega)$ can clearly do this. Analogously, we define $\mathsf{copy}_{\bar{K},\bar{V}}$, but this time we copy keys in $\bar{K}$ to values in $\bar{V}$.

- $\mathsf{add}_{V,0_\oplus}$, for V an attribute-value and $\oplus$ an aggregate operator. Function $\mathsf{add}_{V,0_\oplus}$ takes as input a tuple $t = (\bar{k}, \bar{v}')$ of sort $(\bar{K}, \bar{V}')$, where $V \notin \bar{V}'$, and produces a tuple $t' = (\bar{k}, \bar{v}', 0_\oplus)$ of sort $(\bar{K}, \bar{V}', V)$, i.e., $\mathsf{add}_{V,0_\oplus}$ adds a new value-attribute V that always takes value $0_\oplus$. Analogously, we define functions $\mathsf{add}_{V,\diamondsuit}$ and $\mathsf{add}_{V,\heartsuit}$.
- $\mathsf{eq}_{\bar{K},\bar{K}'}$ and $\mathsf{eq}_{\bar{V},\bar{V}'}$, for $\bar{K}, \bar{K}'$ tuples of key-attributes of the same arity and $\bar{V}, \bar{V}'$ tuples of value-attributes of the same arity. The function $\mathsf{eq}_{\bar{K},\bar{K}'}$ takes as input a tuple $t = (\bar{k}, \bar{v})$ of sort $(\bar{K}_1, \bar{V})$, where $\bar{K}, \bar{K}' \subseteq \bar{K}_1$, and produces as output the tuple $t' = (\bar{k}, \bar{v})$ of sort $(\bar{K}_1, \bar{V})$, if $t(\bar{K}) = t(\bar{K}')$, and the empty associative table otherwise. Hence, this function acts as a filter over an associative table of sort $(\bar{K}_1, \bar{V})$, extending only those tuples t such that $t(\bar{K}) = t(\bar{K}')$. Analogously, we define the function $\mathsf{eq}_{\bar{V},\bar{V}'}$.
- In the same vein, extension functions $\mathsf{neq}_{\bar{K},\bar{K}'}$ and $\mathsf{neq}_{\bar{V},\bar{V}'}$, for $\bar{K}, \bar{K}'$ tuples of key-attributes of the same arity and $\bar{V}, \bar{V}'$ tuples of value-attributes of the same arity. These are defined exactly as $\mathsf{eq}_{\bar{K},\bar{K}'}$ and $\mathsf{eq}_{\bar{V},\bar{V}'}$, only that we now extend only those tuples t such that $t(\bar{K}) \neq t(\bar{K}')$ and $t(\bar{V}) \neq t(\bar{V}')$, respectively.
- The projection $\pi_{\bar{V}}$, for $\bar{V}$ a tuple of value-attributes, takes as input a tuple $(\bar{k}, \bar{v}')$ of sort $(\bar{K}, \bar{V}')$, where $\bar{V} \subseteq \bar{V}'$, and outputs the tuple $(\bar{k}, \bar{v})$ of sort $(\bar{K}, \bar{V})$ such that $\bar{v} = \bar{v}'{\downarrow}_{\bar{V}}$.

We now establish our result.

▶ **Theorem 4.** *Let us assume that Ω contains all extension functions specified above. For every pair $(\phi, \oplus)$, where $\phi(\bar{x}, \bar{\imath})$ is a formula of $\mathrm{FO}^{\mathrm{safe}}_{\mathit{Agg}}(\Psi_\Omega)$ and $\oplus$ is an aggregate operator on Values, there is a $\textsc{Lara}(\Omega)$ expression $e_{\phi,\oplus}[\bar{K}, \bar{V}]$ such that $e^D_{\phi,\oplus} = \phi^D_\oplus = \mathit{Solve}_\oplus(\phi^D)$, for each $\textsc{Lara}$ database D.*

Proof. When f is one of the distinguished extension functions f defined above, we abuse notation and write simply f instead of Ext_f. We first define several useful operations and expressions.

- The projection $\pi^\oplus_{\bar{K}}\, e$ over keys with respect to aggregate operator $\oplus$, defined as $\Join^{\bar{K}}_{\oplus}\, e$. Notice that this removes key-, but not value-attributes from e, i.e., if e is of sort $[\bar{K}', \bar{V}]$ then $\pi^\oplus_{\bar{K}}\, e$ is of sort $[\bar{K}, \bar{V}]$.
- The *rename* operator $\rho_{\bar{K}\to\bar{K}'}\, e$ as $\pi_{\bar{K}'}\, (\mathsf{copy}_{\bar{K},\bar{K}'}\, e)$, where π has no superscript $\oplus$ as no aggregation is necessary in this case. This operation simply renames the key-attributes $\bar{K}$ to a fresh set of key-attributes $\bar{K}'$. Analogously, we define $\rho_{\bar{V}\to\bar{V}'}\, e$, $\rho_{\bar{V}\to\bar{K}}\, e$ and $\rho_{\bar{K}\to\bar{V}}\, e$.
- The *active domain* expression e_{ActDom}, which takes as input a $\textsc{Lara}$ database D and returns all elements $k \in \mathsf{Keys}$ that appear in some fact of D. It is defined as follows. First choose a key attribute not present in any table of D; say it is Z. For each $R[\bar{K}, \bar{V}] \in \sigma$ we define an expression $R^{\mathsf{Keys}} := \pi_\emptyset\, R$, which removes all attribute-values in $\bar{V}$ from R. For each $K \in \bar{K}$ we then define $R^{\mathsf{Keys}}_K := \pi_K\, R^{\mathsf{Keys}}$ as the set of keys that appear in the position of attribute K in $R[\bar{K}, \bar{V}]$ (no need to specify superscript $\oplus$ on π in this case). Finally, we define $e^R_{\mathsf{ActDom}} := \Join_{K\in\bar{K}}\, \rho_{K\to Z} R^{\mathsf{Keys}}_K$ and $e_{\mathsf{ActDom}} := \Join_{R\in\sigma}\, e^R_{\mathsf{ActDom}}$.

We now prove the theorem by induction on ϕ.

- If $\phi = \bot$ then $e_{\phi,\oplus} = \emptyset$ for every aggregate operator $\oplus$.
- If $\phi = (x = y)$, for x, y key-variables, then $e_{\phi,\oplus}[K, K'] := \mathsf{eq}_{K,K'}\big(e_{\mathsf{ActDom}}[K] \bowtie \rho_{K\to K'} e_{\mathsf{ActDom}}[K]\big)$ for every aggregate operator $\oplus$. Notice that there is no need to specify an aggregate operator for $\bowtie$ in this case as the tables that participate in the join only consist of keys.
- Consider now $\phi = R(\bar{x}, \bar{\imath})$, for $R \in \sigma$. We assume all variables in $\bar{x}$ and $\bar{\imath}$, respectively, to be pairwise distinct, as repetition of variables can always be simulated with equalities. Then $e_{\phi,\oplus}[\bar{K}, \bar{V}] := R[\bar{K}, \bar{V}]$ for every aggregate operator $\oplus$.

- Assume that $(\phi, \oplus) = (\phi', \oplus') \wedge \neg(\phi'', \oplus'')$. Let $e_{\phi',\oplus'}[\bar{K}, \bar{V}]$ and $e_{\phi'',\oplus''}[\bar{K}, \bar{V}]$ be the expressions obtained for $(\phi', \oplus')$ and $(\phi'', \oplus'')$, respectively, by induction hypothesis. We construct the expression $e_{\phi,\oplus}$ as follows.
  - First, we take the union of $e_{\phi',\oplus'}$ and $e_{\phi'',\oplus''}$, resolving conflicts with an aggregate operator func that simply checks for which tuples of keys it requires to restore "key-functionality" after performing the union. In particular, func takes as input a multiset of values. If it contains more than one element, it returns the distinguished symbol $\diamondsuit$ which appears in no LARA database D. Otherwise it returns the only element in the multiset. For instance, $\mathsf{func}(\{\!\{a, a\}\!\}) = \diamondsuit$ and $\mathsf{func}(\{\!\{a\}\!\}) = a$.
  Let us define then $e_1[\bar{K}, \bar{V}] := e_{\phi',\oplus'} \,\bowtie\!\!\!\!\!\bowtie_{\mathsf{func}}\, e_{\phi'',\oplus''}$. Notice that e_1^D, for D a LARA database, contains the tuples $(\bar{k}, \bar{v}) \in e^D_{\phi',\oplus'}$ such that there is no tuple of the form $(\bar{k}, \bar{v}') \in e^D_{\phi'',\oplus''}$, plus the tuples of the form $(\bar{k}, \diamondsuit, \dots, \diamondsuit)$ such that there are tuples of the form $(\bar{k}, \bar{v}') \in e^D_{\phi',\oplus'}$ and $(\bar{k}, \bar{v}'') \in e^D_{\phi'',\oplus''}$. In other words, by evaluating e_1 on D we have marked with $(\diamondsuit, \dots, \diamondsuit)$ those tuples $\bar{k}$ of keys that are candidates to be removed when computing the difference $e_{\phi',\oplus'} \setminus e_{\phi'',\oplus''}$.
  - Second, we take the join $e_{\phi',\oplus'}[\bar{K}, \bar{V}] \bowtie e_1'[\bar{K}, \bar{V}']$, where $e_1'[\bar{K}, \bar{V}'] := \rho_{\bar{V} \to \bar{V}'} e_1[\bar{K}, \bar{V}]$ is obtained by simply renaming $\bar{V}$ as $\bar{V}'$ in e_1 (there is no need to specify an aggregate operator for $\bowtie$ as $\bar{V}$ and $\bar{V}'$ have no attributes in common), and apply the extension function $\mathsf{eq}_{\bar{V},\bar{V}'}$ over it. It is easy to see that when evaluating the resulting expression $e_\alpha[\bar{K}, \bar{V}] := \mathsf{eq}_{\bar{V},\bar{V}'}(e_{\phi',\oplus'} \bowtie e_1')$ on a LARA database D, we obtain precisely the tuples $(\bar{k}, \bar{v}) \in e^D_{\phi',\otimes'}$ such that there is no tuple of the form $(\bar{k}, \bar{w}) \in e^D_{\phi'',\otimes''}$. In fact, for those tuples we also have that $(\bar{k}, \bar{v})$ belongs to e_1, as explained above, and thus $\mathsf{eq}_{\bar{V},\bar{V}'}$ applies no filter. On the contrary, if there is a tuple of the form $(\bar{k}, \bar{w}) \in e^D_{\phi'',\oplus''}$ then $(\bar{k}, \diamondsuit, \dots, \diamondsuit) \in e_1^D$ as explained above. This means that the filter $\mathsf{eq}_{\bar{V},\bar{V}'}$ is applied and no tuple of the form $(\bar{k}, \dots)$ appears in the result. (Notice that for the latter to hold we use, in an essential way, the "key-functionality" of tables $e^D_{\phi',\oplus'}$ and $e^D_{\phi'',\oplus''}$ and the fact that $\diamondsuit$ does not appear in D). By definition, then, $e^D_\alpha \subseteq e^D_{\phi,\oplus}$.
  - Third, we take the join $e_{\phi',\oplus'}[\bar{K}, \bar{V}] \bowtie e'_{\phi'',\oplus''}[\bar{K}, \bar{V}']$, where
  $$e'_{\phi'',\oplus''}[\bar{K}, \bar{V}'] := \rho_{\bar{V} \to \bar{V}'} e_{\phi'',\oplus''}[\bar{K}, \bar{V}]$$
  is obtained by simply renaming $\bar{V}$ as $\bar{V}'$ in $e_{\phi'',\oplus''}$ (there is no need to specify an aggregate operator for $\bowtie$ as $\bar{V}$ and $\bar{V}'$ have no attributes in common), and apply the extension function $\mathsf{neq}_{\bar{V},\bar{V}'}$ over it. It is easy to see that when evaluating the resulting expression $e_\beta[\bar{K}, \bar{V}] := \pi_{\bar{V}} \mathsf{neq}_{\bar{V},\bar{V}'}(e_{\phi',\oplus'} \bowtie e'_{\phi'',\oplus''})$ on a LARA database D, we obtain precisely the tuples $(\bar{k}, \bar{v}) \in e^D_{\phi',\otimes'}$ such that there is a tuple of the form $(\bar{k}, \bar{w}) \in e^D_{\phi'',\oplus''}$ for which $\bar{v} \neq \bar{w}$. This means that the evaluation of e_β on D contains precisely the tuples $(\bar{k}, \bar{v}) \in e^D_{\phi',\oplus'}$ that do not belong to e^D_α, yet they belong to $e^D_{\phi',\oplus'} \setminus e^D_{\phi'',\oplus''}$. (Notice that for the latter to hold we use, in an essential way, the "key-functionality" of tables $e^D_{\phi',\oplus'}$ and $e^D_{\phi'',\oplus''}$).
  - Summing up, we can now define $e_{\phi,\oplus}[\bar{K}, \bar{V}]$ as $e_\alpha[\bar{K}, \bar{V}] \,\bowtie\!\!\!\!\!\bowtie\, e_\beta[\bar{K}, \bar{V}]$. Notice that there is no need to specify an aggregate operator for $\bowtie\!\!\!\!\!\bowtie$ here, as by construction we have that there are no tuples $\bar{k}$ of keys that belong to both $\pi_{\bar{K}} e^D_\alpha$ and $\pi_{\bar{K}} e^D_\beta$.
- Assume that $(\phi, \oplus) = (\phi', \oplus') \wedge (\phi'', \oplus'')$. Let $e_{\phi',\oplus'}[\bar{K}, \bar{V}]$ and $e_{\phi'',\oplus''}[\bar{K}, \bar{V}]$ be the expressions obtained for $(\phi', \oplus')$ and $(\phi'', \oplus'')$, respectively, by induction hypothesis. We construct the expression $e_{\phi,\oplus}$ by taking the join $e_{\phi',\oplus'}[\bar{K}, \bar{V}] \bowtie e'_{\phi'',\oplus''}[\bar{K}, \bar{V}']$, where
$$e'_{\phi'',\oplus''}[\bar{K}, \bar{V}'] := \rho_{\bar{V} \to \bar{V}'} e_{\phi'',\oplus''}[\bar{K}, \bar{V}]$$

is obtained by simply renaming $\bar{V}$ as $\bar{V}'$ in $e_{\phi'',\oplus''}$ (there is no need to specify an aggregate operator for $\bowtie$ as $\bar{V}$ and $\bar{V}'$ have no attributes in common), and apply $\pi_{\bar{V}}\mathsf{eq}_{\bar{V},\bar{V}'}$ over it. In fact, if a tuple $(\bar{k},\bar{v})$ is selected by this expression it means that $(\bar{k},\bar{v}) \in e^D_{\phi',\oplus'} \cap e^D_{\phi'',\oplus''}$ by definition of $\mathsf{eq}_{\bar{V},\bar{V}'}$. In turn, if $(\bar{k},\bar{v})$ is not selected by the expression it means either that there is no tuple of the form $(\bar{k},\bar{w}) \in e^D_{\phi'',\oplus''}$, or the unique tuple of the form $(\bar{k},\bar{w}) \in e^D_{\phi'',\oplus''}$ satisfies that $\bar{v} \neq \bar{w}$ (due to the way in which $\mathsf{eq}_{\bar{V},\bar{V}'}$ is defined). In any of the two cases we have that $(\bar{k},\bar{v}) \notin e^D_{\phi',\oplus'} \cap e^D_{\phi'',\oplus''}$.

- Assume that $(\phi,\oplus) = (\phi',\oplus') \vee (\phi'',\oplus'')$. Let $e_{\phi',\oplus'}[\bar{K},\bar{V}]$ and $e_{\phi'',\oplus''}[\bar{K},\bar{V}]$ be the expressions obtained for $(\phi',\oplus')$ and $(\phi'',\oplus'')$, respectively, by induction hypothesis. We construct the expression $e_{\phi,\oplus}$ by taking the union of the following expressions.
  - An expression e_1 such that, when evaluated on a LARA database D, it computes the tuples $(\bar{k},\bar{v}) \in e^D_{\phi',\oplus'}$ for which there is no tuple of the form $(\bar{k},\bar{w}) \in e^D_{\phi'',\oplus''}$. This can be done in the same way as we constructed e_α for the case when $(\phi,\oplus) = (\phi',\oplus') \wedge \neg(\phi'',\oplus'')$ (see above).
  - Analogously, an expression e_2 that computes the tuples $(\bar{k},\bar{v}) \in e^D_{\phi'',\oplus''}$ for which there is no tuple of the form $(\bar{k},\bar{w}) \in e^D_{\phi',\oplus'}$.
  - An expression e_3 such that, when evaluated on a LARA database D, it computes the tuples $(\bar{k},\bar{v}) \in e^D_{\phi',\oplus'}$ for which there is a tuple of the form $(\bar{k},\bar{w}) \in e^D_{\phi'',\oplus''}$ that satisfies $\bar{v} = \bar{w}$. This can be done in the same way as we did in the previous point.
  - An expression e_4 such that, when evaluated on a LARA database D, it computes the tuples $(\bar{k},\bar{v})$ that are of the form $(\bar{k},\bar{w}_1 \oplus \bar{w}_2)$ for $(\bar{k},\bar{w}_1) \in e^D_{\phi',\oplus'}$ and $(\bar{k},\bar{w}_2) \in e^D_{\phi'',\oplus''}$ with $\bar{w}_1 \neq \bar{w}_2$. The expression e_4 can be defined as $e_\alpha \mathbin{\rtimes\!\!\!\ltimes} e_\beta$, where e^D_α contains all tuples $(\bar{k},\bar{v}) \in e^D_{\phi',\oplus'}$ such that there is a tuple of the form $(\bar{k},\bar{w}) \in e^D_{\phi'',\oplus''}$ that satisfies $\bar{v} \neq \bar{w}$, and e^D_β contains all tuples $(\bar{k},\bar{v}) \in e^D_{\phi'',\oplus''}$ such that there is a tuple of the form $(\bar{k},\bar{w}) \in e^D_{\phi',\oplus'}$ that satisfies $\bar{v} \neq \bar{w}$. It is easy to see how to express e_α and e_β by using techniques similar to the ones developed in the previous points.
- Assume that $(\phi,\oplus) = (\phi',\oplus') \wedge k = \tau(\bar{x},\bar{i})$, for a formula $\phi'(\bar{x},\bar{i})$ and a value-term τ of $\mathrm{FO}^{\mathrm{safe}}_{\mathsf{Agg}}(\Psi_\Omega)$, and k a value-variable not necessarily present in $\bar{i}$. We only consider the case when k is not in $\bar{i}$. The other case is similar. Before we proceed we prove the following lemma which is basic for the construction (the proof is omitted due to lack of space).

  ▶ **Lemma 5.** *For every pair $(\alpha,\oplus_\alpha)$, where $\alpha(\bar{x},\bar{i})$ is a formula of $\mathrm{FO}^{\mathrm{safe}}_{\mathsf{Agg}}(\Psi_\Omega)$ and $\oplus_\alpha$ is an aggregate operator over* Values*, and for every value-term $\lambda(\bar{x},\bar{i})$ of $\mathrm{FO}^{\mathrm{safe}}_{\mathsf{Agg}}(\Psi_\Omega)$, there is an expression $e_{\alpha,\oplus_\alpha,\lambda}[\bar{K},\bar{V},V_1]$ of LARA(Ω) such that for every LARA database D:*

  $$(\bar{k},\bar{v},v_1) \in e^D_{\alpha,\oplus_\alpha,\lambda} \iff \big((\bar{k},\bar{v}) \in \alpha^D_{\oplus_\alpha} \text{ and } \lambda(\bar{k},\bar{v}) = v_1\big).$$

  It should be clear then that $e_{\phi,\oplus} = e_{\phi',\oplus',\tau}[\bar{K},\bar{V},V_1]$, where $e_{\phi',\oplus',\tau}[\bar{K},\bar{V},V_1]$ is the expression constructed for $(\phi',\oplus')$ and τ by applying Lemma 5.
- Assume that $(\phi,\oplus) = (\phi',\oplus') \wedge R_f(\bar{x},\bar{x}',\bar{i},\bar{i}')$, where $\phi(\bar{x},\bar{i})$ is a formula of $\mathrm{FO}^{\mathrm{safe}}_{\mathsf{Agg}}(\Psi_\Omega)$. Let $e_{\phi',\oplus'}[\bar{K},\bar{V}]$ be the expression obtained for $(\phi',\oplus')$ by induction hypothesis. We can then define the expression $e_{\phi,\oplus}$ as $\mathsf{Ext}_f(e_{\phi',\oplus'}[\bar{K},\bar{V}]) \bowtie e_{\phi',\oplus'}[\bar{K},\bar{V}]$, assuming that f is of sort $(\bar{K},\bar{V}) \to (\bar{K}',\bar{V}')$. There is no need to specify an aggregate operator for $\bowtie$ here, since by assumption we have that $\bar{V} \cap \bar{V}' = \emptyset$.
- The cases $(\phi,\oplus) = \exists x(\phi',\oplus')$ and $(\phi,\oplus) = \exists i(\phi',\oplus')$ can be translated as $\pi^{\oplus}_{\bar{K}}\, e_{\phi',\oplus'}[K,\bar{K},\bar{V}]$ and $\pi_{\bar{V}}\, e_{\phi',\oplus'}[\bar{K},\bar{V},V]$, respectively, assuming that $e_{\phi',\oplus'}$ is the expression obtained for $(\phi',\oplus')$ by induction hypothesis.

This finishes the proof of the theorem. ◀

Discussion

The results presented in this section imply that LARA has the same expressive power as $\mathrm{FO}_{\mathsf{Agg}}$, which in turn is tightly related to the expressiveness of SQL [13]. One might wonder then why to use LARA instead of SQL. While it is difficult to give a definite answer to this question, we would like to note that LARA is especially tailored to deal with ML objects, such as matrices or tensors, which are naturally modeled as associative tables. As the proof of Theorem 4 suggests, in turn, $\mathrm{FO}_{\mathsf{Agg}}$ requires of several cumbersome tricks to maintain the "key-functionality" of associative tables.

5 Expressiveness of LARA in terms of ML Operators

We assume in this section that $\mathsf{Values} = \mathbb{Q}$. Since extension functions in Ω can a priori be arbitrary, to understand what LARA can express we first need to specify which classes of functions are allowed in Ω. In rough terms, this is determined by the operations that one can perform when comparing keys and values, respectively. We explain this below.

- Extensions of two-sorted logics with aggregate operators over a *numerical* sort $\mathcal{N}$ often permit to perform arbitrary numerical comparisons over $\mathcal{N}$ (in our case $\mathcal{N} = \mathsf{Values} = \mathbb{Q}$). It has been noted that this extends the expressive power of the language, while at the same time preserving some properties of the logic that allow to carry out an analysis of its expressiveness based on well-established techniques (see, e.g., [14]).
- In some cases in which the expressive power of the language needs to be further extended, one can also define a linear order on the non-numerical sort (which in our case is the set Keys) and then perform suitable arithmetic comparisons in terms of such a linear order. A well-known application of this idea is in the area of descriptive complexity [11].

We start in this section by considering the first possibility only. That is, we allow comparing elements of $\mathsf{Values} = \mathbb{Q}$ in terms of arbitrary numerical operations. Elements of Keys, in turn, can only be compared with respect to equality. This yields a logic that is amenable for theoretical exploration – in particular, in terms of its expressive power – and that at the same time is able to express many extension functions of practical interest (e.g., several of the functions used in examples in [9, 10]).

We design a simple logic $\mathrm{FO}(=, \mathsf{All})$ for expressing extension functions. Intuitively, the name of this logic states that it can only compare keys with respect to equality but it can compare values in terms of arbitrary numerical predicates. The formulas in the logic are standard FO formulas where the only atomic expressions allowed are of the following form:

- $x = y$, for x, y key-variables;
- $P(i_1, \dots, i_k)$, for $P \subseteq \mathbb{Q}^k$ a numerical relation of arity k and $i_1, \dots, i_k$ value-variables or constants of the form $0_\oplus$.

The semantics of this logic is standard. In particular, an assignment η from value-variables to $\mathbb{Q}$ *satisfies* a formula of the form $P(i_1, \dots, i_k)$, for $P \subseteq \mathbb{Q}^k$, whenever $\eta(i_1, \dots, i_k) \in P$.

Let $\phi(\bar{x}, \bar{y}, \bar{i}, \bar{j})$ be a formula of $\mathrm{FO}(=, \mathsf{All})$. For a tuple $t = (\bar{k}, \bar{k}', \bar{v}, \bar{v}') \in \mathsf{Keys}^{|\bar{k}|+|\bar{k}'|} \times \mathsf{Values}^{|\bar{v}|+|\bar{v}'|}$ we abuse terminology and say that $\phi(\bar{k}, \bar{k}', \bar{v}, \bar{v}')$ *holds* if $D_t \models \phi(\bar{k}, \bar{k}', \bar{v}, \bar{v}')$, where D_t is the database composed exclusively by tuple t. In addition, an extension function f of sort $(\bar{K}, \bar{V}) \mapsto (\bar{K}', \bar{V}')$ is *definable* in $\mathrm{FO}(=, \mathsf{All})$, if there is a formula $\phi_f(\bar{x}, \bar{y}, \bar{i}, \bar{j})$ of $\mathrm{FO}(=, \mathsf{All})$, for $|\bar{x}| = |\bar{K}|$, $|\bar{y}| = |\bar{K}'|$, $|\bar{i}| = |\bar{V}|$, and $|\bar{j}| = |\bar{V}'|$, such that for every tuple $(\bar{k}, \bar{v})$ of sort $(\bar{K}, \bar{V})$ it is the case

$$f(\bar{k}, \bar{v}) \;=\; \{(\bar{k}', \bar{v}') \mid \phi(\bar{k}, \bar{k}', \bar{v}, \bar{v}') \text{ holds}\}.$$

This gives rise to the definition of the following class of extension functions:

$$\Omega_{(=,\mathsf{All})} \;=\; \{f \mid f \text{ is an extension function that is definable in FO}(=,\mathsf{All})\}.$$

Recall that extension functions only produce finite associative tables by definition, and hence only some formulas in FO$(=,\mathsf{All})$ define extension functions.

The extension functions $\mathsf{copy}_{\bar{K},\bar{K}'}$, $\mathsf{copy}_{\bar{V},\bar{V}'}$, $\mathsf{add}_{V,0_\oplus}$, $\mathsf{eq}_{\bar{K},\bar{K}'}$, $\mathsf{eq}_{\bar{V},\bar{V}'}$, $\mathsf{neq}_{\bar{K},\bar{K}'}$, $\mathsf{neq}_{\bar{V},\bar{V}'}$, and $\pi_{\bar{V}}$, as shown in the previous section, are in $\Omega_{(=,\mathsf{All})}$. In turn, $\mathsf{copy}_{\bar{V},\bar{K}}$ and $\mathsf{copy}_{\bar{K},\bar{V}}$ are not, as FO$(=,\mathsf{All})$ cannot compare keys with values. Next we provide more examples.

▶ **Example 6.** We use $i+j=k$ and $ij=k$ as a shorthand notation for the ternary numerical predicates of addition and multiplication, respectively. Consider first a function f that takes a tuple t of sort (K_1,K_2,V) and computes a tuple t' of sort (K_1',K_2',V') such that $t(K_1,K_2)=t'(K_1',K_2')$ and $t'(V')=1-t(V)$. Then f is definable in FO$(=,\mathsf{All})$ as $\phi_f(x,y,x',y',i,j) := \big(x=x' \wedge y=y' \wedge i+j=1\big)$. This function can be used, e.g., to interchange 0s and 1s in a Boolean matrix.

Consider now a function g that takes a tuple t of sort (K,V_1,V_2) and computes a tuple t' of sort (K',V') such that $t(K)=t'(K')$ and $t'(V)$ is the average between $t(V_1)$ and $t(V_2)$. Then g is definable in FO$(=,\mathsf{All})$ as $\phi_g(x,y,i_1,i_2,j) := \big(x=y \wedge \exists i\,(i_1+i_2=i \wedge 2j=i)\big)$. ◀

As an immediate corollary to Theorem 3 we obtain the following result, which formalizes the fact that – in the case when $\mathsf{Values}=\mathbb{Q}$ – for translating LARA$(\Omega_{(=,\mathsf{All})})$ expressions it is not necessary to extend the expressive power of FO$_{\mathsf{Agg}}$ with the relations in $\Psi_{\Omega_{(=,\mathsf{All})}}$ as long as one has access to all numerical predicates over $\mathbb{Q}$. Formally, let us denote by FO$_{\mathsf{Agg}}(\mathsf{All})$ the extension of FO$_{\mathsf{Agg}}$ with all formulas of the form $P(\iota_1,\ldots,\iota_k)$, for $P\subseteq\mathbb{Q}^k$ and $\iota_1,\ldots,\iota_k$ value-terms, with the expected semantics. Then one can prove the following result.

▶ **Corollary 7.** *For every expression $e[\bar{K},\bar{V}]$ of LARA$(\Omega_{(=,All)})$ there is a formula $\phi_e(\bar{x},\bar{\imath})$ of FO$_{Agg}(All)$ such that $e^D=\phi_e^D$, for every LARA database D.*

It is known that queries definable in FO$_{\mathsf{Agg}}(\mathsf{All})$ satisfy two important properties, namely, *genericity* and *locality*, which allow us to prove that neither convolution of matrices nor matrix inversion can be defined in the language. From Corollary 7 we obtain then that none of these queries is expressible in LARA$(\Omega_{(=,\mathsf{All})})$. We explain this next.

Convolution

Let A be an arbitrary matrix and K a square matrix. For simplicity we assume that K is of odd size $(2n+1)\times(2n+1)$. The convolution of A and K, denoted by $A*K$, is a matrix of the same size as A whose entries are defined as

$$(A*K)_{k\ell} \;=\; \sum_{s=1}^{2n+1}\sum_{t=1}^{2n+1} A_{k-n+s,\ell-n+t}\cdot K_{st}. \tag{6}$$

Notice that $k-n+s$ and $\ell-n+t$ could be invalid indices for matrix A. The standard way of dealing with this issue is *zero padding*. This simply assumes those entries outside A to be 0. In the context of the convolution operator, one usually calls K a *kernel*.

We represent A and K over the schema $\sigma=\{\mathsf{Entry}_A[K_1,K_2,V],\mathsf{Entry}_K[K_1,K_2,V]\}$. Assume that $\mathsf{Keys}=\{\mathsf{k}_1,\mathsf{k}_2,\mathsf{k}_3,\ldots\}$ and $\mathsf{Values}=\mathbb{Q}$. If A is a matrix of values in $\mathbb{Q}$ of dimension $m\times p$, and K is a matrix of values in $\mathbb{Q}$ of dimensions $(2n+1)\times(2n+1)$ with $m,p,n\geq 1$, we represent the pair (A,K) as the LARA database $D_{A,K}$ over σ that contains all facts $\mathsf{Entry}_A(\mathsf{k}_i,\mathsf{k}_j,A_{ij})$, for $i\in[m]$, $j\in[p]$, and all facts $\mathsf{Entry}_K(\mathsf{k}_i,\mathsf{k}_j,K_{ij})$, for $i\in[2n+1]$,

$j \in [2n+1]$. The query Convolution over schema σ takes as input a LARA database of the form $D_{A,K}$ and returns as output an associative table of sort $[K_1, K_2, V]$ that contains exactly the tuples $(\mathsf{k}_i, \mathsf{k}_j, (A * K)_{ij})$. We can then prove the following result.

▶ **Proposition 8.** *Convolution is not expressible in* LARA$(\Omega_{(=,\mathsf{All})})$.

The proof is based on a simple *genericity* property for the language that is not preserved by convolution.

Matrix inverse

It has been shown by Brijder et al. [1] that matrix inversion is not expressible in MATLANG by applying techniques based on locality. The basic idea is that MATLANG is subsumed by $\mathrm{FO}_{\mathsf{Agg}}(\emptyset) = \mathrm{FO}_{\mathsf{Agg}}$, and the latter logic can only define *local* properties. Intuitively, this means that formulas in $\mathrm{FO}_{\mathsf{Agg}}$ can only distinguish up to a *fixed-radius* neighborhood from its free variables (see, e.g., [14] for a formal definition). On the other hand, as shown in [1], if matrix inversion were expressible in MATLANG there would also be a $\mathrm{FO}_{\mathsf{Agg}}$ formula that defines the transitive closure of a binary relation (represented by its adjacency Boolean matrix). This is a contradiction as transitive closure is the prime example of a non-local property. We use the same kind of techniques to show that matrix inversion is not expressible in LARA$(\Omega_{(=,\mathsf{All})})$. For this, we use the fact that $\mathrm{FO}_{\mathsf{Agg}}(\mathsf{All})$ is also local.

We represent Boolean matrices as databases over the schema $\sigma = \{\mathsf{Entry}[K_1, K_2, V]\}$. Assume that $\mathsf{Keys} = \mathbb{N}$ and $\mathsf{Values} = \mathbb{Q}$. The Boolean matrix M of dimension $n \times m$, for $n, m \geq 1$, is represented as the LARA database D_M over σ that contains all facts $\mathsf{Entry}(i, j, b_{ij})$, for $i \in [n]$, $j \in [m]$, and $b_{ij} \in \{0, 1\}$, such that $M_{ij} = b_{ij}$. Consider the query Inv over schema σ that takes as input a LARA database of the form D_M and returns as output the LARA database $D_{M^{-1}}$, for M^{-1} the inverse of M. Then:

▶ **Proposition 9.** *LARA$(\Omega_{(=,\mathsf{All})})$ cannot express Inv over Boolean matrices. That is, there is no LARA$(\Omega_{(=,\mathsf{All})})$ expression $e_{\mathsf{Inv}}[K_1, K_2, V]$ over σ such that $e_{\mathsf{Inv}}(D_M) = \mathsf{Inv}(D_M)$, for every LARA database of the form D_M that represents a Boolean matrix M.*

6 Adding Built-in Predicates over Keys

In Section 5 we have seen that there are important linear algebra operations, such as matrix inverse and convolution, that LARA$(\Omega_{(=,\mathsf{All})})$ cannot express. The following result shows, on the other hand, that a clean extension of LARA$(\Omega_{(=,\mathsf{All})})$ can express matrix convolution. This extension corresponds to the language LARA$(\Omega_{(<,\mathsf{All})})$, i.e., the extension of LARA$(\Omega_{(=,\mathsf{All})})$ in which we assume the existence of a strict linear-order $<$ on Keys and extension functions are definable in the logic $\mathrm{FO}(<, \mathsf{All})$ that extends $\mathrm{FO}(=, \mathsf{All})$ by allowing atomic formulas of the form $x < y$, for x, y key-variables. Even more, the only numerical predicates from All we need are $+$ and $\times$. We denote the resulting logic as LARA$(\Omega_{(<,\{+,\times\})})$.

▶ **Proposition 10.** *CONVOLUTION is expressible in LARA$(\Omega_{(<,\{+,\times\})})$.*

It is worth remarking that Hutchison et al. [9] showed that for every fixed kernel K, the query $(A * K)$ is expressible in LARA. However, the LARA expression they construct depends on the values of K, and hence their construction does not show that in general convolution is expressible in LARA. Our construction is stronger, as we show that there exists a *fixed* LARA$(\Omega_{(<,\{+,\times\})})$ expression that takes A and K as input and produces $(A * K)$ as output.

Current ML libraries usually have specific implementations for the convolution operator. Although specific implementations can lead to very efficient ways of implementing a single convolution, they could prevent the optimization of pipelines that merge several convolutions with other operators. Proposition 10 shows that convolution can be expressed in LARA just using general abstract operators such as aggregation and filtering. This could open the possibility for optimizing expressions that mix convolution and other operators.

Can LARA($\Omega_{(<,\{+,\times\})}$) express inverse?

We believe that LARA($\Omega_{(<,\{+,\times\})}$) cannot express INV. However, this seems quite challenging to prove. First, the tool we used for showing that INV is not expressible in LARA($\Omega_{(=,\mathsf{All})}$), namely, locality, is no longer valid in this setting. In fact, queries expressible in LARA($\Omega_{(<,\{+,\times\})}$) are not necessarily local.

▶ **Proposition 11.** *LARA($\Omega_{(<,\{+,\times\})}$) can express non-local queries.*

This implies that one would have to apply techniques more specifically tailored for the logic, such as *Ehrenfeucht-Fraïssé* games, to show that INV is not expressible in LARA($\Omega_{(<,\{+,\times\})}$). Unfortunately, it is often combinatorially difficult to apply such techniques in the presence of built-in predicates, e.g., a linear order, on the domain; cf., [5, 17, 7]. So far, we have not managed to succeed in this regard.

On the other hand, we can show that INV is not expressible in a natural restriction of LARA($\Omega_{(<,\{+,\times\})}$) under complexity-theoretic assumptions. To start with, INV is complete for the complexity class DET, which contains all those problems that are logspace reducible to computing the *determinant* of a matrix. It is known that LOGSPACE $\subseteq$ DET, where LOGSPACE is the class of functions computable in logarithmic space, and this inclusion is believed to be proper [3].

In turn, most of the aggregate operators used in practical applications, including standard ones such as SUM, AVG, MIN, MAX, and COUNT, can be computed in LOGSPACE (see, e.g., [2]). Combining this with well-known results on the complexity of computing relational algebra and arithmetic operations, we obtain that the fragment LARA$_{\mathrm{st}}$($\Omega_{(<,\{+,\times\})}$) of LARA($\Omega_{(<,\{+,\times\})}$) that only mentions the standard aggregate operators above, and whose formulas defining extension functions are *safe*, can be evaluated in LOGSPACE in *data complexity*, i.e., assuming formulas to be fixed.

▶ **Proposition 12.** *Let $e[\bar{K}, \bar{V}]$ be a fixed expression of LARA$_{\mathrm{st}}$($\Omega_{(<,\{+,\times\})}$). There is a LOGSPACE procedure that takes as input a LARA database D and computes e^D.*

Hence, proving INV to be expressible in the language LARA$_{\mathrm{st}}$($\Omega_{(<,\{+,\times\})}$) would imply the surprising result that LOGSPACE = DET.

7 Final Remarks and Future Work

We believe that the work on query languages for analytics systems that integrate relational and statistical functionalities provides interesting perspectives for database theory. In this paper we focused on the LARA language, which has been designed to become the core algebraic language for such systems. As we have observed, expressing interesting ML operators in LARA requires the addition of complex features, such as arithmetic predicates on the numerical sort and built-in predicates on the domain. The presence of such features complicates the study of the expressive power of the languages, as some known techniques no longer hold, e.g., genericity and locality, while others become combinatorially difficult to apply, e.g., Ehrenfeucht-Fraïssé games. In addition, the presence of a built-in linear order might turn the logic capable of characterizing some parallel complexity classes, and thus inexpressibility results could be as hard to prove as some longstanding conjectures in complexity theory.

A possible way to overcome these problems might be not looking at languages in its full generality, but only at extensions of the tame fragment LARA($\Omega_{(=,\mathsf{All})}$) with some of the most sophisticated operators. For instance, what if we extend LARA($\Omega_{(=,\mathsf{All})}$) directly with an operator that computes Convolution? Is it possible to prove that the resulting language (LARA($\Omega_{(=,\mathsf{All})}$) + Convolution) cannot express matrix inverse Inv? Somewhat a similar approach has been followed in the study of MATLANG; e.g., [1] studies the language (MATLANG + INV), which extends MATLANG with the matrix inverse operator.

Another interesting line of work corresponds to identifying which kind of operations need to be added to LARA in order to be able to express in a natural way recursive operations such as matrix inverse. One would like to do this in a general yet minimalistic way, as adding too much recursive expressive power to the language might render it impractical. It would be important to start then by identifying the most important recursive operations one needs to perform on associative tables, and then abstract from them the minimal primitives that the language needs to possess for expressing such operations.

References

1 Robert Brijder, Floris Geerts, Jan Van den Bussche, and Timmy Weerwag. On the Expressive Power of Query Languages for Matrices. In *ICDT*, pages 10:1–10:17, 2018.

2 Mariano P. Consens and Alberto O. Mendelzon. Low Complexity Aggregation in GraphLog and Datalog. *Theor. Comput. Sci.*, 116(1):95–116, 1993.

3 Stephen A. Cook. A Taxonomy of Problems with Fast Parallel Algorithms. *Information and Control*, 64(1-3):2–21, 1985.

4 Serge Abiteboul et al. Research Directions for Principles of Data Management (Dagstuhl Perspectives Workshop 16151). *Dagstuhl Manifestos*, 7(1):1–29, 2018.

5 Ronald Fagin, Larry J. Stockmeyer, and Moshe Y. Vardi. On Monadic NP vs. Monadic co-NP. *Inf. Comput.*, 120(1):78–92, 1995.

6 Floris Geerts. On the Expressive Power of Linear Algebra on Graphs. In *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal*, pages 7:1–7:19, 2019.

7 Martin Grohe and Thomas Schwentick. Locality of Order-Invariant First-Order Formulas. In *MFCS*, pages 437–445, 1998.

8 Stephan Hoyer, Joe Hamman, and xarray developers. xarray Development roadmap, 2018. Available at `http://xarray.pydata.org/en/stable/roadmap.html`, retrieved on March 2019.

9 Dylan Hutchison, Bill Howe, and Dan Suciu. Lara: A Key-Value Algebra underlying Arrays and Relations. *CoRR*, abs/1604.03607, 2016.

10 Dylan Hutchison, Bill Howe, and Dan Suciu. LaraDB: A Minimalist Kernel for Linear and Relational Algebra Computation. In *Proceedings of BeyondMR@SIGMOD 2017*, pages 2:1–2:10, 2017.

11 Neil Immerman. *Descriptive complexity.* Graduate texts in computer science. Springer, 1999.

12 Andreas Kunft, Alexander Alexandrov, Asterios Katsifodimos, and Volker Markl. Bridging the gap: towards optimization across linear and relational algebra. In *Proceedings of BeyondMR@SIGMOD 2016*, page 1, 2016.

13 Leonid Libkin. Expressive power of SQL. *Theor. Comput. Sci.*, 296(3):379–404, 2003.

14 Leonid Libkin. *Elements of Finite Model Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

15 Alexander M. Rush. Tensor Considered Harmful. Technical report, Harvard NLP Blog, 2019. Available at `http://nlp.seas.harvard.edu/NamedTensor`, retrieved on March 2019.

16 Alexander M. Rush. Tensor Considered Harmful Pt. 2. Technical report, Harvard NLP Blog, 2019. Available at `http://nlp.seas.harvard.edu/NamedTensor2`, retrieved on March 2019.

17 Thomas Schwentick. On Winning Ehrenfeucht Games and Monadic NP. *Ann. Pure Appl. Logic*, 79(1):61–92, 1996.