TriAL for RDF: Adapting Graph Query Languages for RDF Data

Leonid Libkin
University of Edinburgh
libkin@ed.ac.uk

Juan Reutter
University of Edinburgh and PUC Chile
juan.reutter@ed.ac.uk

Domagoj Vrgoč
University of Edinburgh
domagoj.vrgoc@ed.ac.uk

ABSTRACT

Querying RDF data is viewed as one of the main applications of graph query languages, and yet the standard model of graph databases – essentially labeled graphs – is different from the triples-based model of RDF. While encodings of RDF databases into graph data exist, we show that even the most natural ones are bound to lose some functionality when used in conjunction with graph query languages. The solution is to work directly with triples, but then many properties taken for granted in the graph database context (e.g., reachability) lose their natural meaning.

Our goal is to introduce languages that work directly over triples and are closed, i.e., they produce sets of triples, rather than graphs. Our basic language is called TriAL, or Triple Algebra: it guarantees closure properties by replacing the product with a family of join operations. We extend TriAL with recursion, and explain why such an extension is more intricate for triples than for graphs. We present a declarative language, namely a fragment of datalog, capturing the recursive algebra. For both languages, the combined complexity of query evaluation is given by low-degree polynomials. We compare our languages with relational languages, such as finite-variable logics, and previously studied graph query languages such as adaptations of XPath, regular path queries, and nested regular expressions; many of these languages are subsumed by the recursive triple algebra. We also provide examples of the usefulness of TriAL in querying graph and RDF data.

Categories and Subject Descriptors. F.4.1 [**Mathematical logic and formal languages**]: Mathematical logic; H.2.1 [**Database Management**]: Logical Design—*Data Models*; H.2.3 [**Database management**]: Languages—*Query Languages*

Keywords. RDF, Triple Algebra, Query evaluation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PODS'13, June 22–27, 2013, New York, New York, USA.
Copyright 2013 ACM 978-1-4503-2066-5/13/06 ...$15.00.

1. INTRODUCTION

Graph data management is currently one of the most active research topics in the database community, fueled by the adoption of graph models in new application domains, such as social networks, bioinformatics and astronomic databases, and projects such as the Web of Data and the Semantic Web. There are many proposals for graph query languages; we now understand many issues related to query evaluation over graphs, and there are multiple vendors offering graph database products, see [2, 3, 14, 37] for surveys.

The Semantic Web and its underlying data model, RDF, are usually cited as one of the key applications of graph databases, but there is some mismatch between them. The standard model of graph databases [2, 37] that dates back to [12, 13], is that of directed edge-labeled graphs, i.e., pairs $G = (V, E)$, where V is a set of vertices (objects), and E is a set of labeled edges. Each labeled edge is of the form (v, a, v'), where v, v' are nodes in V, and a is a label from some finite labeling alphabet Σ. As such, they are the same as labeled transition systems used as a basic model in both hardware and software verification. Graph databases of course can store data associated with their nodes (e.g., information about each person in a social network).

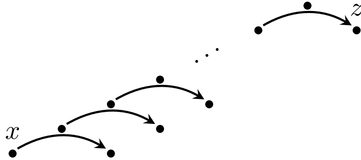
The model of RDF data is very similar, yet slightly different. The basic concept is a *triple* (s, p, o), that consists of the subject s, the predicate p, and the object o, drawn from a domain of uniform resource identifiers (URI's). Thus, the middle element need not come from a finite alphabet, and may in addition play the role of a subject or an object in another triple. For instance, $\{(s, p, o), (p, s, o')\}$ is a valid set of RDF triples, but in graph databases, it is impossible to have two such edges.

To understand why this mismatch is a problem, consider querying graph data. Since graph databases and RDF are represented as relations, relational queries can be applied to them. But crucially, we may also query the *topology* of a graph. For instance, many graph query languages have, as their basic building block, *regular path queries*, or RPQs [13], that find nodes reachable by a path whose label belongs to a regular language.

We take the notion of reachability for granted in graph databases, but what is the corresponding notion for triples, where the middle element can serve as the source and the target of an edge? Then there are multiple possibilities, two of which are illustrated below.

Query $\mathsf{Reach}_{\rightarrow}$ looks for pairs (x, z) connected by paths of the following shape:

and $\mathsf{Reach}_{\nearrow}$ looks for the following connection pattern:



But can such patterns be defined by existing RDF query languages? Or can they be defined by existing graph query languages under some graph encoding of RDF?

To answer these, we need to understand which navigational facilities are available for RDF data. A recent survey of graph database systems [3] shows that, by and large, they either offer support for triples, or they do graphs and then can express proper reachability queries. An attempt to add navigation to RDF languages was made in [33], where a language called nSPARQL was defined by taking SPARQL [22, 32], the standard query language for RDF, and extending it with a navigational mechanism provided by *nested regular expressions*. These are essentially regular path queries with XPath-inspired node tests. The evaluation of those uses essentially a graph encoding of RDF. As the starting point of our investigation, we show that there are natural reachability patterns for triples, similar to those shown above, that *cannot* be defined in graph encodings of RDF [5] using nested regular expressions, nor in nSPARQL itself.

Thus, navigational patterns over triples are beyond reach of both RDF languages and graph query languages that work on encodings of RDF. The solution is then to design languages that work directly on RDF triples, and have both relational and navigational querying facilities, just like graph query languages. Our goal, therefore, is to adapt graph database techniques for direct RDF querying.

A crucial property of a query language is *closure*: queries should return objects of the same kind as their input. Closed languages, therefore, are compositional: their operators can be applied to results of queries. Using graph languages for RDF suffers from non-compositionality: for instance, RPQs return graphs rather than triples. So we start by defining a closed language for triples. To understand its basic operations, we first look at a language that has essentially first-order expressivity, and then add navigational features.

We take relational algebra as the basic language. Clearly projection violates closure so we throw it away. Selection and set operations, on the other hand, are fine. The problematic operation is Cartesian product: if T, T' are sets of triples, then $T \times T'$ is not a set of triples but rather a set of 6-tuples. What do we do then? We shall need reachability in the language, and for graphs, reachability is computed by iterating *composition* of relations. The composition operation for binary relations preserves closure: a pair (x, y) is in the composition $R \circ R'$ of R and R' iff $(x, z) \in R$ and $(z, y) \in R'$ for some z. So this is a join of R and R' and it seems that what we need is it analog for triples.

But queries $\mathsf{Reach}_{\rightarrow}$ and $\mathsf{Reach}_{\nearrow}$ demonstrate that there is no such thing as *the reachability* for triples. In fact, we shall see that there is not even a nice analog of composition for triples. So instead, we add *all* possible joins that keep the algebra closed. The resulting language is called *Triple Algebra*, denoted by TriAL. We then add an iteration mechanism to it, to enable it to express reachability queries based on different joins, and obtain *Recursive Triple Algebra* TriAL^*.

The algebra TriAL^* can express both reachability patterns above, as well as queries we prove to be inexpressible in nSPARQL. It has a declarative language associated with it, a fragment of Datalog. It has good query evaluation bounds: combined complexity is (low-degree) polynomial. Moreover, we exhibit a fragment with complexity of the order $O(|e| \cdot |O| \cdot |T|)$, where e is the query, O is the set of objects in the database, and T is the set of triples. This is a very natural fragment, as it restricts arbitrary recursive definitions to those essentially defining reachability properties.

The model we use is slightly more general than just triples of objects and amounts to combining triplestores as in, e.g., [24] with the representation of objects used in the Neo4j database [14, 31]. Each object participating in a triple comes associated with a set of attributes. Of course this can be modeled via more triples, but the model we use is conceptually cleaner and leads to a more natural comparison with other query languages.

The first of those comparisons is with relational querying. We show that TriAL lives between FO^3 and FO^6 (recall that FO^k refers to the fragment of First-Order Logic using only k variables). In fact it contains FO^3, is contained in FO^6, and is incomparable with FO^4 and FO^5. A similar result holds for TriAL^* and transitive closure logic.

On the graph querying side, we show that the navigational power of TriAL^* subsumes that of both regular path queries and nested regular expressions. In fact it subsumes a version of *graph XPath* recently proposed for graph databases [27]. We also compare it with conjunctive RPQs [12] and some of their extensions studied

in [10, 11]. When it comes to graphs with data held in their nodes, we show that TriAL^* continues to subsume some of the formalisms proposed in that context, such as graph XPath expanded with data tests and some types of regular expressions with data values [28, 27].

This shows that TriAL^* is an expressive language that subsumes a number of well known relational and graph formalisms, that permits navigational queries not expressible on graph encodings of RDF or in nSPARQL, and that has good query evaluation properties.

Organization In Section 2 we review graph and RDF databases, and describe our model. We also show that some natural navigational queries over triples cannot be expressed in languages such as nSPARQL. In Section 3 we define TriAL and TriAL^* and study their expressiveness. In Section 4 we give a declarative language capturing TriAL^*. In Section 5 we study query evaluation, and in Sections 6.1 and 6.2 we study our languages in connection with relational and graph querying.

2. GRAPH DATABASES AND RDF

Basic Definitions

Graph databases. We now review some standard definitions (see, e.g., [2, 11, 37]). A graph database is just a finite edge-labeled graph in which each node has a data value attached. Formally, let $\mathcal{N}$ be a countably infinite set of *node ids*, Σ a finite alphabet and $\mathcal{D}$ a countably infinite set of data values. Then a *graph database* over Σ is a triple $G = (V, E, \rho)$, where $V \subset \mathcal{N}$ is a finite set of nodes, $E \subseteq V \times \Sigma \times V$ is a set of labeled edges, and $\rho : V \to \mathcal{D}$ is a function assigning a data value to each node. Each edge is a triple (u, a, v), whose interpretation is an a-labeled edge from u to v. When Σ is clear from the context, we shall simply speak of a graph database. If we work with graph databases that make no use of data values, we write $G = (V, E)$ and disregard the function ρ.

A *path* π from u_0 to u_m in G is a sequence (u_0, a_0, u_1), $(u_1, a_1, u_2), \cdots, (u_{m-1}, a_{m-1}, u_m)$, where each (u_i, a_i, u_{i+1}), for $i < m$, is an edge in E. The *label* of π, denoted by $\lambda(\pi)$, is the word $a_0 \cdots a_{m-1} \in \Sigma^*$.

Regular path queries. Typical navigational languages for graph databases use *regular path queries*, or *RPQ*s [13] as the basic building block. An RPQ is an expression $x \xrightarrow{L} y$, where x and y are variables and L is a regular language over Σ. Given a graph database $G = (V, E)$ over Σ, it defines pairs of nodes (u, v) such that there is a path π from u to v with $\lambda(\pi) \in L$.

Nested regular expressions. These expressions, abbreviated as NRE, over a finite alphabet Σ, extend ordinary regular expressions with the nesting operator (essentially the node test of XPath) and inverses [8, 33]. Formally they are defined as follows:

$$e := \varepsilon \mid a \mid a^- \mid e \cdot e \mid e^* \mid e + e \mid [e], \quad a \in \Sigma.$$

An NRE defines, over a graph $G = (V, E)$, a binary relation on V. The semantics of ε is the diagonal $\{(u, u) \mid u \in V\}$; the semantics of a is the set $\{(u, v) \mid (u, a, v) \in E\}$ of a-labeled edges, and a^- defines $\{(u, v) \mid (v, a, u) \in E\}$. Operations $\cdot$, $+$, and $*$ denote composition, union, and transitive closure of binary relations. Finally, the node test $[e]$ defines pairs (u, u) so that (u, v) is in the result of e for some $v \in V$.

RDF databases. RDF databases contain triples in which, unlike in graph databases, the middle component need not come from a fixed set of labels. Formally, if $\mathbf{U}$ is a countably infinite domain of uniform resource identifiers (URI's), then an RDF triple is $(s, p, o) \in \mathbf{U} \times \mathbf{U} \times \mathbf{U}$, where s is referred to as the subject, p as the predicate, and o as the object. An RDF graph is just a collection of RDF triples. Here we deal with *ground* RDF documents [33], i.e., we do not consider blank nodes or literals in RDF documents (otherwise we need to deal with disjoint domains, which complicates the presentation).

Example 1. The RDF database D in Figure 1 contains information about cities, modes of transportation between them, and operators of those services. Each triple is represented by an arrow from the subject to the object, with the arrow itself labeled with the predicate. Examples of triples in D are (`Edinburgh`, `Train Op 1`, `London`) and (`Train Op 1`, `part_of`, `EastCoast`). For simplicity, we assume from now on that we can determine implicitly whether an object is a city or an operator. This can of course be modeled by adding an additional outgoing edge labeled `city` from each city and `operator` from each service operator.

Graph Queries for RDF

Navigational properties (e.g., reachability patterns) are among the most important functionalities of RDF query languages. However, typical RDF query languages, such as SPARQL, are in spirit relational languages. To extend them with navigation, as in [33, 4, 30], one typically uses features inspired by graph query languages, surveyed briefly earlier. Nonetheless, such approaches have their inherent limitations, as we explain here.

Looking again at the database D in Figure 1, we see the main difference between graphs and RDF: the majority of the edge labels in D are also used as subjects or objects (i.e., nodes) of other triples of D. For instance, one can travel from Edinburgh to London by using a train service Train Op 1, but in this case the label itself is viewed as a node when we express the fact that this operator is actually a part of EastCoast trains.

For RDF, one normally uses a model of *triplestores* that is different from graph databases. According to it, the database from Figure 1 is viewed as a ternary relation:

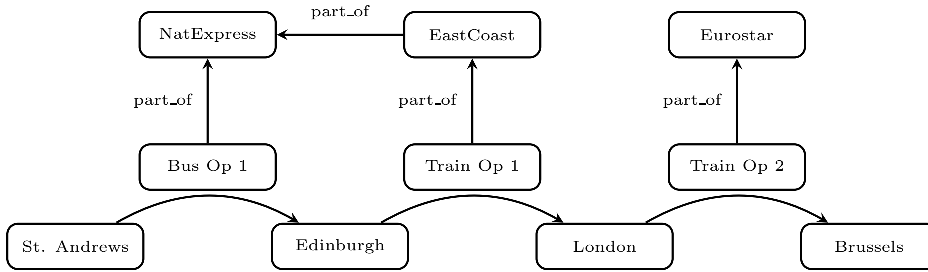


Figure 1: RDF graph storing information about cities and transport services between them

St. Andrews	Bus Op 1	Edinburgh
Edinburgh	Train Op 1	London
London	Train Op 2	Brussels
Bus Op 1	part_of	NatExpress
Train Op 1	part_of	EastCoast
Train Op 2	part_of	Eurostar
EastCoast	part_of	NatExpress

Suppose one wants to answer the following query:

Q : *Find pairs of cities (x, y) such that one can travel from x to y using services operated by the same company.*

A query like this is likely to be relevant, for instance, when integrating numerous transport services into a single ticketing interface. In our example, the pair (Edinburgh, London) belongs to $Q(D)$, and one can also check that (St. Andrews, London) is in $Q(D)$, since recursively both operators are part of NatExpress (using the transitivity of part_of). However, the pair (St. Andrews, Brussels) does not belong to $Q(D)$, since we can only travel that route if we change companies, from NatExpress to Eurostar.

To enhance SPARQL with navigational properties, [33] added nested regular expressions to it, resulting in a language called nSPARQL. The idea was to combine the usual reachability patterns of graph query languages with the XPath mechanism of node tests. However, nested regular expressions, which we saw earlier, are defined for graphs, and not for databases storing triples. Thus, they cannot be used directly over RDF databases; instead, one needs to transform an RDF database D into a graph first. An example of such transformation $D \to \sigma(D)$ was given in [5]; it is illustrated in Figure 2.

Formally, given an RDF document D, the graph $\sigma(D) = (V, E)$ is a graph database over alphabet $\Sigma = \{\text{next}, \text{node}, \text{edge}\}$, where V contains all resources from D, and for each triple (s, p, o) in D, the edge relation E contains edges (s, edge, p), (p, node, o) and (s, next, o). This transformation scheme is important in practical RDF applications (it was shown to be crucial for addressing the problem of interpreting RDFS features within SPARQL [33]). At the same time, it is not sufficient for expressing simple reachability patterns like those in query Q:

Proposition 1. *The query Q is not expressible by NREs over graph transformations $\sigma(\cdot)$ of ternary relations.*

Thus, the most common RDF navigational mechanism cannot express a very natural property, essentially due to the need to do so via a graph transformation.

One might argue that this result is due to the shortcomings of a specific transformation (however relevant to practical tasks it might be). So we ask what happens in the native RDF scenario. In particular, we would like to see what happens with the language nSPARQL [33], which is a proper RDF query language extending SPARQL with navigation based on nested regular expressions. But this language falls short too, as it fails to express the simple reachability query Q.

Theorem 1. *The query Q above cannot be expressed in nSPARQL.*

The key reason for these limitations is that the navigation mechanisms used in RDF languages are graph-based, when one really needs them to be triple-based.

Triplestore Databases

To introduce proper triple-based navigational languages, we first define a simple model of triplestores. Let $\mathcal{O}$ be a countably infinite set of objects, and $\mathcal{D}$ be a countably infinite set of data values.

Definition 1. *A* triplestore database*, or just* triplestore *over $\mathcal{D}$ is a tuple $T = (O, E_1, \ldots, E_n, \rho)$, where:*

- *$O \subset \mathcal{O}$ is a finite set of objects,*
- *each $E_i \subseteq O \times O \times O$ is a set of triples, and*
- *$\rho : O \to \mathcal{D}$ is a function that assigns a data value to each object.*

Often we have just a single ternary relation E in a triplestore database (e.g., in the previously seen examples of representing RDF databases), but all the languages and results we state here apply to multiple relations. The function ρ could also map $\mathcal{O}$ to tuples over $\mathcal{D}$, and all results remain true (one just uses $\mathcal{D}^k$ as the range of ρ, as in the example below). We use the function $\rho : O \to \mathcal{D}$ just to simplify notations.

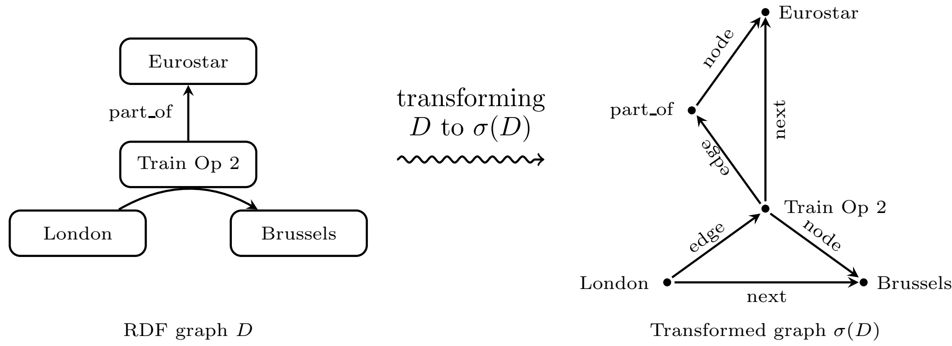


Figure 2: Transforming part of the RDF database from Figure 1 into a graph database

Triplestores easily model RDF, and we will see later that they model graph databases. Furthermore, they can be used to model several other applications relying on semistructured data, such as e.g. social networks.

3. AN ALGEBRA FOR RDF

We saw that problems encountered while adapting graph languages to RDF are related to the inherent limitations of the graph data model for representing RDF data. Thus, one should work directly with triples. But existing languages are either based on binary relations and fall short of the power necessary for RDF querying, or are general relational languages which are not closed when it comes to querying RDF triples. Hence, we need a language that works directly on triples, is closed, and has good query evaluation properties.

We now present such a language, based on relational algebra for triples. We start with a plain version and then add recursive primitives that provide the crucial functionality for handling reachability properties.

The operations of the usual relational algebra are selection, projection, union, difference, and cartesian product. Our language must remain *closed*, i.e., the result of each operation ought to be a valid triplestore. This clearly rules out projection. Selection and Boolean operations are fine. Cartesian product, however, would create a relation of arity six, but instead we use *joins* that only keep three positions in the result.

Triple joins. To see what kind of joins we need, let us first look at the *composition* of two relations. For binary relations S and S', their composition $S \circ S'$ has all pairs (x, y) so that $(x, z) \in S$ and $(z, y) \in S'$ for some z. Reachability with relation S is defined by recursively applying composition: $S \cup S \circ S \cup S \circ S \circ S \cup \ldots$. So we need an analog of composition for triples. To understand how it may look, we can view $S \circ S'$ as the *join* of S and S' on the condition that the 2nd component of S equals the first of S', and the output consist of the remaining components. We can write it as

$$S \overset{1,2'}{\underset{2=1'}{\bowtie}} S'$$

Here we refer to the positions in S as 1 and 2, and to the positions in S' as $1'$ and $2'$, so the join condition is $2 = 1'$ (written below the join symbol), and the output has positions 1 and $2'$. This suggests that our join operations on triples should be of the form $R \bowtie^{i,j,k}_{\text{cond}} R'$, where R and R' are tertiary relations, $i, j, k \in \{1, 2, 3, 1', 2', 3'\}$, and cond is a condition (to be defined precisely later).

But what is the most natural analog of relational composition? Note that to keep three indexes among $\{1, 2, 3, 1', 2', 3'\}$, we ought to project away three, meaning that two of them will come from one argument, and one from the other. Any such join operation on triples is bound to be *asymmetric*, and thus cannot be viewed as a full analog of relational composition.

So what do we do? Our solution is to add *all* such join operations. Formally, given two tertiary relations R and R', *join* operations are of the form

$$R \overset{i,j,k}{\underset{\theta,\eta}{\bowtie}} R',$$

where

- $i, j, k \in \{1, 1', 2, 2', 3, 3'\}$,
- θ is a set of equalities and inequalities between elements in $\{1, 1', 2, 2', 3, 3'\} \cup \mathcal{O}$,
- η is a set of equalities and inequalities between elements in $\{\rho(1), \rho(1'), \rho(2), \rho(2'), \rho(3), \rho(3')\} \cup \mathcal{D}$.

The semantics is defined as follows: (o_i, o_j, o_k) is in the result of the join iff there are triples $(o_1, o_2, o_3) \in R$ and $(o_{1'}, o_{2'}, o_{3'}) \in R'$ such that

- each condition from θ holds; that is, if $l = m$ is in θ, then $o_l = o_m$, and if $l = o$, where o is an object, is in θ, then $o_l = o$, and likewise for inequalities;
- each condition from η holds; that is, if $\rho(l) = \rho(m)$ is in η, then $\rho(o_l) = \rho(o_m)$, and if $\rho(l) = d$, where d is a data value, is in η, then $\rho(o_l) = d$, and likewise for inequalities.

Triple Algebra. We now define the expressions of the *Triple Algebra*, or TriAL for short. It is a restriction

of relational algebra that guarantees closure, i.e., the result of each expression is a triplestore.

- Every relation name in a triplestore is a TriAL expression.
- If e is a TriAL expression, θ a set of equalities and inequalities over $\{1,2,3\} \cup \mathcal{O}$, and η is a set of equalities and inequalities over $\{\rho(1), \rho(2), \rho(3)\} \cup \mathcal{D}$, then $\sigma_{\theta,\eta}(e)$ is a TriAL expression.
- If e_1, e_2 are TriAL expressions, then the following are TriAL expressions:
  - $e_1 \cup e_2$;
  - $e_1 - e_2$;
  - $e_1 \Join^{i,j,k}_{\theta,\eta} e_2$, where i, j, k, θ, η as in the definition of the join above.

The semantics of the join operation has already been defined. The semantics of the Boolean operations is the usual one. The semantics of the selection is defined in the same way as the semantics of the join (in fact, the operator itself can be defined in terms of joins): one just chooses triples (o_1, o_2, o_3) satisfying both θ and η.

Given a triplestore database T, we write $e(T)$ for the result of expression e on T.

Note that $e(T)$ is again a triplestore, and thus TriAL defines closed operations on triplestores. This is important, for instance, when we require RDF queries to produce RDF graphs as their result (instead of arbitrary tuples of objects), as it is done in SPARQL via the CONSTRUCT operator [34].

Example 2. To get some intuition about the Triple Algebra consider the following TriAL expression:

$$e = E \underset{2=1'}{\overset{1,3',3}{\Join}} E$$

Indexes $(1,2,3)$ refer to positions of the first triple, and indexes $(1',2',3')$ to positions of the second triple in the join. Thus, for two triples (x_1, x_2, x_3) and $(x_{1'}, x_{2'}, x_{3'})$, such that $x_2 = x_{1'}$, expression e outputs the triple $(x_1, x_{3'}, x_3)$. E.g., in the triplestore of Fig. 1, (`London`, `Train Op 2`, `Brussels`) is joined with (`Train Op 2`, `part_of`, `Eurostar`), producing (`London`, `Eurostar`, `Brussels`); the full result is

St. Andrews	NatExpress	Edinburgh
Edinburgh	EastCoast	London
London	Eurostar	Brussels

Thus, e computes travel information for pairs of European cities together with companies one can use. It fails to take into account that `EastCoast` is a part of `NatExpress`. To add such information to query results (and produce triples such as (`Edinburgh`, `NatExpress`, `London`)), we use $e' = e \cup (e \Join^{1,3',3}_{2=1'} E)$.

Definable operations: intersection and complement. As usual, the intersection operation can be defined as $e_1 \cap e_2 = e_1 \Join^{1,2,3}_{1=1',2=2',3=3'} e_2$. Note that using join and union, we can define the set U of all triples (o_1, o_2, o_3) so that each o_i occurs in our triplestore database T. For instance, to collect all such triples so that o_1 occurs in the first position of R, and o_2, o_3 occur in the 2nd and 3rd position of R' respectively, we would use the expression $(R \Join^{1,2',3} R') \Join^{1,2,3'} R'$. Taking the union of all such expressions, gives us the relation U.

Using such U, we can define e^c, the complement of e with respect to the active domain, as $U - e$. In what follows, we regularly use intersection and complement in our examples.

Adding Recursion. One problem with Example 2 above is that it does not include triples ($\texttt{city}_1$,`service`,$\texttt{city}_2$) so that relation R contains a triple ($\texttt{city}_1$,$\texttt{service}_0$,$\texttt{city}_2$), and there is a chain, of some length, indicating that $\texttt{service}_0$ is a part of `service`. The second expression in Example 2 only accounted for such paths of length 1. To deal with paths of arbitrary length, we need reachability, which relational algebra is well known to be incapable of expressing. Thus, we need to add recursion to our language.

To do so, we expand TriAL with *right* and *left Kleene closure* of any triple join $\Join^{i,j,k}_{\theta,\eta}$ over an expression e, denoted as $(e \Join^{i,j,k}_{\theta,\eta})^*$ for right, and $(\Join^{i,j,k}_{\theta,\eta} e)^*$ for left. These are defined as

$$\begin{aligned}(e \Join)^* &= \emptyset \cup e \cup e \Join e \cup (e \Join e) \Join e \cup \ldots, \\ (\Join e)^* &= \emptyset \cup e \cup e \Join e \cup e \Join (e \Join e) \cup \ldots\end{aligned}$$

We refer to the resulting algebra as *Triple Algebra with Recursion* and denote it by TriAL*.

When dealing with binary relations we do not have to distinguish between left and right Kleene closures, since the composition operation for binary relations is associative. However, as the following example shows, joins over triples are not necessarily associative, which explains the need to make this distinction.

Example 3. Consider a triplestore database $T = (O, E)$, with $E = \{(a,b,c), (c,d,e), (d,e,f)\}$. The function ρ is not relevant for this example. The expression

$$e_1 = (E \underset{3=1'}{\overset{1,2,2'}{\Join}})^*$$

computes $e_1(T) = E \cup \{(a,b,d), (a,b,e)\}$, while

$$e_2 = (\underset{3=1'}{\overset{1,2,2'}{\Join}} E)^*$$

computes $e_2(T) = E \cup \{(a,b,d)\}$.

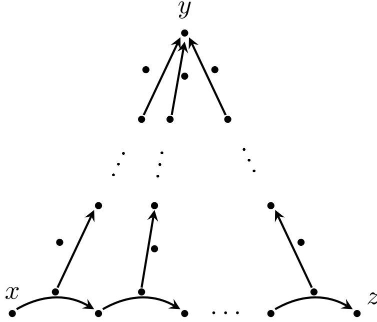
Now we present several examples of queries one can ask using the Triple Algebra.

Example 4. We refer now to reachability queries $\mathsf{Reach}_{\to}$ and $\mathsf{Reach}_{\nearrow}$ from the introduction. It can easily be checked that these are defined by

$$(E \underset{3=1'}{\overset{1,2,3'}{\bowtie}})^* \quad \text{and} \quad (\underset{1=2'}{\overset{1',2',3}{\bowtie}} E)^*$$

respectively.

Next consider the query from Theorem 1. Graphically, it can be represented as follows:



That is, we are looking for pairs of cities such that one can travel from one to the other using services operated by the same company. This query is expressed by

$$((E \underset{2=1'}{\overset{1,3',3}{\bowtie}})^* \underset{3=1',2=2'}{\overset{1,2,3'}{\bowtie}})^*.$$

Note that the interior join $(E \underset{2=1'}{\overset{1,3',3}{\bowtie}})^*$ computes all triples (x, y, z), such that $E(x, w, z)$ holds for some w, and y is reachable from w using some E-path. The outer join now simply computes the transitive closure of this relation, taking into account that the service that witnesses the connection between the cities is the same.

4. A DECLARATIVE LANGUAGE

Triple Algebra and its recursive versions are *procedural* languages. In databases, we are used to dealing with declarative languages. The most common one for expressing queries that need recursion is Datalog. It is one of the most studied database query languages, and it has reappeared recently in numerous applications. One instance of this is its well documented success in Web information extraction [19] and there are numerous others. So it seems natural to look for Datalog fragments to capture TriAL and its recursive version.

Since Datalog works over relational vocabularies, we need to explain how to represent triplestores T. The schema of these representations consists of a ternary relation symbol $E(\cdot, \cdot, \cdot)$ for each triplestore name in T, plus a binary relation symbol $\sim(\cdot, \cdot)$. Each triplestore database T can be represented as an instance I_T of this schema in the standard way: the interpretation of each relation name E in this instance corresponds to the triples in the triplestore E in T, and the interpretation of $\sim$ contains all pairs (x, y) of objects such that $\rho(x) = \rho(y)$, i.e. x and y have the same data value. If the values of ρ are tuples, we just use $\sim_i$ relations testing that the ith components of tuples are the same, for each i; this does not affect the results here at all.

We start with a Datalog fragment capturing TriAL. A TripleDatalog rule is of the form

$$\begin{aligned} S(\bar{x}) \ &\leftarrow \ S_1(\bar{x}_1), S_2(\bar{x}_2), \\ &\sim(y_1, z_1), \ldots, \sim(y_n, z_n), u_1 = v_1, \ldots, u_m = v_m \quad (1) \end{aligned}$$

where

1. S, S_1 and S_2 are (not necessarily distinct) predicate symbols of arity at most 3;
2. all variables in $\bar{x}$ and each of y_i, z_i and u_j, v_j are contained in $\bar{x}_1$ or $\bar{x}_2$.

A $\mathsf{TripleDatalog}^\neg$ rule is like the rule (1) but all equalities and predicates, except the head predicate S, can appear negated. A $\mathsf{TripleDatalog}^\neg$ *program* Π is a finite set of $\mathsf{TripleDatalog}^\neg$ rules. Such a program Π is *nonrecursive* if there is an ordering $r_1, \ldots, r_k$ of the rules of Π so that the relation in the head of r_i does not occur in the body of any of the rules r_j, with $j \leq i$.

As is common with non-recursive programs, the semantics of nonrecursive $\mathsf{TripleDatalog}^\neg$ programs is given by evaluating each of the rules of Π, according to the order $r_1, \ldots, r_k$ of its rules, and taking unions whenever two rules have the same relation in their head (see [1] for the precise definition). We are now ready to present the first capturing result.

Proposition 2. TriAL *is equivalent to nonrecursive* $\mathsf{TripleDatalog}^\neg$ *programs.*

We next turn to the expressive power of recursive Triple Algebra TriAL^*. To capture it, we of course add recursion to Datalog rules, and impose a restriction that was previously used in [12]. A $\mathsf{ReachTripleDatalog}^\neg$ *program* is a $\mathsf{TripleDatalog}^\neg$ program in which each recursive predicate S is the head of exactly two rules of the form:

$$\begin{aligned} S(\bar{x}) \ &\leftarrow \ R(\bar{x}) \\ S(\bar{x}) \ &\leftarrow \ S(\bar{x}_1), R(\bar{x}_2), V(y_1, z_1), \ldots, V(y_k, z_k) \end{aligned}$$

where each $V(y_i, z_i)$ is one of the following: $y_i = z_i$, or $y_i \neq z_i$, or $\sim(y_i, z_i)$, or $\neg\sim(y_i, z_i)$, and R is a nonrecursive predicate of arity at most 3. These rules essentially mimic the standard reachability rules (for binary relation) in Datalog, and in addition one can impose equality and inequality constraints, as well as data equality and inequality constraints, along the paths.

Note that the negation in $\mathsf{ReachTripleDatalog}^\neg$ programs is *stratified*. The semantics of these programs is the standard least-fixpoint semantics [1]. A similarly defined syntactic class, but over graph databases, rather than triplestores, was shown to capture the expressive power of FO with the transitive closure operator [12]. In our case, we have a capturing result for TriAL^*.

Theorem 2. *The expressive power of* TriAL* *and* ReachTripleDatalog$^\neg$ *programs is the same.*

Next we give an example of a simple datalog program computing the query from Theorem 1.

Example 5. The following ReachTripleDatalog$^\neg$ program is equivalent to query Q from Theorem 1. Note that the answer is computed in the predicate Ans.

$$\begin{array}{rcl} S(x_1, x_2, x_3) & \leftarrow & E(x_1, x_2, x_3) \\ S(x_1, x_3', x_3) & \leftarrow & S(x_1, x_2, x_3), E(x_2, x_2', x_3') \\ \text{Ans}(x_1, x_2, x_3) & \leftarrow & S(x_1, x_2, x_3) \\ \text{Ans}(x_1, x_2, x_3') & \leftarrow & \text{Ans}(x_1, x_2, x_3), S(x_3, x_2, x_3') \end{array}$$

Recall that this query can be written in TriAL* as $Q = ((E \bowtie^{1,3',3}_{2=1'})^* \bowtie^{1,2,3'}_{3=1',2=2'})^*$. The predicate S in the program computes the inner Kleene closure of the query, while the predicate Ans computes the outer closure.

5. QUERY EVALUATION

In this section we analyze two versions of the query evaluation problems related to Triple Algebra. The *query evaluation* problem is to check if a given tuple is in the result of a query (as is standard in the study of complexity of database queries, especially when one wants to know which complexity classes they belong to). The query *computation* problem is to produce the output $e(T)$ for an expression e and a triplestore database T. We start with query evaluation.

Problem:	QueryEvaluation
Input:	A TriAL* expression e, a triplestore T and a tuple (x_1, x_2, x_3) of objects.
Question:	Is $(x_1, x_2, x_3) \in e(T)$?

Many graph query languages (e.g., RPQs) have Ptime upper bounds for this problem, and the data complexity (i.e., when e is assumed to be fixed) is generally in NLogspace (which cannot be improved, since the simplest reachability problem over graphs is already NLogspace-hard). We now show that the same upper bounds hold for our algebra, even with recursion.

Proposition 3. *The problem* QueryEvaluation *is* Ptime*-complete, and in* NLogspace *if the algebra expression e is fixed.*

Tractable evaluation (even with respect to combined complexity) is practically a must when dealing with very large and dynamic semi-structured databases. However, in order to make a case for the practical applicability of our algebra, we need to give more precise bounds for query evaluation, rather than describe complexity classes the problem belongs to. We now show that TriAL* expressions can be evaluated in what is essentially cubic time with respect to the data. Thus, in the rest of the section we focus on the problem of actually computing the whole relation $e(T)$:

Problem:	QueryComputation
Input:	A TriAL* expression e and a triplestore database T.
Output:	The relation $e(T)$

We now analyze the complexity of QueryComputation. Following an assumption frequently made in papers on graph database query evaluation (in particular, graph pattern matching algorithms) as well as bounded variable relational languages (cf. [16, 15, 20]), we consider an *array representation* for triplestores. That is, when representing a triplestore $T = (O, E_1, \ldots, E_m, \rho)$ with $O = \{o_1, \ldots, o_n\}$, we assume that each relation E_l is given by a three-dimensional $n \times n \times n$ matrix, so that the ijkth entry is set to 1 iff (o_i, o_j, o_k) is in E_l. Alternatively we can have a single matrix, where entries include sets of indexes of relations E_l that triples belong to. Furthermore we have a one-dimensional array of size n whose ith entry contains $\rho(o_i)$. Using this representation we obtain the following bounds.

Theorem 3. *The problem* QueryComputation *can be solved in time*

- $O(|e| \cdot |T|^2)$ *for* TriAL *expressions,*
- $O(|e| \cdot |T|^3)$ *for* TriAL* *expressions.*

Note that this immediately gives the Ptime upper bound for Proposition 3.

One can examine the proofs of Proposition 2 and Theorem 2 and see that translations from Datalog into algebra are linear-time. Thus, we have the same bound for the query computation problem, when we evaluate a Datalog program Π in place of an algebra expression.

Corollary 1. *The problem* QueryComputation *for Datalog programs Π can be solved in time*

- $O(|\Pi| \cdot |T|^2)$ *for* TripleDatalog$^\neg$ *programs,*
- $O(|\Pi| \cdot |T|^3)$ *for* ReachTripleDatalog$^\neg$ *programs.*

Lower-complexity fragments. Even though we have acceptable combined complexity of query computation, if the size of T is very large, one may prefer to lower the it even further. We now look at fragments of TriAL* for which this is possible.

In algorithms from Theorem 3, the main difficulty arises from the presence of inequalities in join conditions. A natural restriction then is to look at a fragment TriAL$^=$ of TriAL in which all conditions θ and η used in joins can only use equalities. This fragment allows us to lower the $|T|^2$ complexity, by replacing one of the $|T|$ factors by $|O|$, the number of distinct objects.

Proposition 4. *The* QueryComputation *problem for* TriAL$^=$ *expressions can be solved in time $O(|e| \cdot |O| \cdot |T|)$.*

To pose navigational queries, one needs the recursive algebra, so the question is whether similar bounds can

be obtained for meaningful fragments of TriAL^*. Using the ideas from the proof of Theorem 3 we immediately get an $O(|e| \cdot |O| \cdot |T|^2)$ upper bound for $\mathsf{TriAL}^=$ with recursion. However, we can improve this result for the fragment $\mathsf{reachTA}^=$ that extends $\mathsf{TriAL}^=$ with essentially *reachability* properties, such as those used in RPQs and similar query languages for graph databases.

To define it, we restrict the star operator to mimic the following graph database reachability queries:

- the query "reachable by an arbitrary path", expressed by $(R \bowtie^{1,2,3'}_{3=1'})^*$; and
- the query "reachable by a path labeled with the same element", expressed by $(R \bowtie^{1,2,3'}_{3=1',2=2'})^*$.

These are the only applications of the Kleene star permitted in $\mathsf{reachTA}^=$. For this fragment, we have the same lower complexity bound.

Proposition 5. *The problem* QUERYCOMPUTATION *for* $\mathsf{reachTA}^=$ *can be solved in time* $O(|e| \cdot |O| \cdot |T|)$.

6. TRIPLE ALGEBRA AND OTHER LANGUAGES

In this section we compare the expressive power of our algebras with relational and graph languages. As usual, we say that a language $\mathcal{L}_1$ is contained in a language $\mathcal{L}_2$ if for every query in $\mathcal{L}_1$ there is an equivalent query in $\mathcal{L}_2$. If in addition $\mathcal{L}_2$ has a query not expressible in $\mathcal{L}_1$, then $\mathcal{L}_1$ is strictly contained in $\mathcal{L}_2$. The languages are equivalent if each is contained in the other. They are incomparable if none is contained in the other.

6.1 Triple Algebra as a Relational language

To compare TriAL with relational languages, we use exactly the same relational representation of triplestores as we did when we found Datalog fragments capturing TriAL and TriAL^*. That is, we compare the expressive power of TriAL with that of First–Order Logic (FO) over vocabulary $\langle E_1, \ldots, E_n, \sim \rangle$.

Since TriAL is a restriction of relational algebra, of course it is contained in FO. We do a more detailed analysis based on the number of variables. Recall that FO^k stands for FO restricted to k variables only. To give an intuition why such restrictions are relevant for us, consider, for instance, the join operation $e = E \bowtie^{1,3',3}_{2=2'} E$. It can be expressed by the following FO^6 formula: $\varphi(x_1, x_{3'}, x_3) = \exists x_2 \exists x_{1'} \exists x_{2'} \big(E(x_1, x_2, x_3) \wedge E(x_{1'}, x_{2'}, x_{3'}) \wedge x_2 = x_{2'}\big)$. This suggests that we can simulate joins using only six variables, and this extends rather easily to the whole algebra. One can furthermore show that the containment is proper in this case.

What about fragments of FO using fewer variables? Clearly we cannot go below three variables. It is not difficult to show that TriAL simulates FO^3, but the relationship with the 4 and 5 variable formalisms appears much more intricate, and its study requires more involved techniques. We can show the following.

Theorem 4.

- TriAL *is strictly contained in* FO^6.
- FO^3 *is strictly contained in* TriAL.
- TriAL *is incomparable with* FO^4 *and* FO^5.

The containment of FO^3 in TriAL is proved by induction, and we use pebble games to show that such containment is proper. For the last, more involved part of the theorem, we first show that TriAL is not contained in FO^5. Notice that the expression e given by

$$U \bowtie^{1,2,3}_{\theta} U, \text{ with } \theta = \{i \neq j \mid i, j \in \{1, 1', 2, 2', 3, 3'\}, i < j\},$$

is such that $e(T)$ is not empty if and only if T has six different objects (recall that U is the set of all triples (o_1, o_2, o_3) so that each o_i occurs in a triple in T). It then follows that TriAL is not contained in FO^5 (nor FO^4), cf. [26]. To show that FO^4 is not contained in TriAL, we devise a game that characterizes expressibility of TriAL, and use this game to show that TriAL cannot express the following FO^4 query $\varphi(x, y, z)$:

$$\exists w \big(\psi(x, y, w) \wedge \psi(x, w, z) \wedge \psi(w, y, z) \wedge \psi(x, y, z)\big),$$

where

$$\psi(x, y, z) = \exists w \big(E(x, w, y) \wedge E(y, w, z) \wedge E(z, w, x)\big).$$

The above result also shows that TriAL cannot express all conjunctive queries, since in particular the query $\varphi(x, y, z)$ is a conjunctive query. This is of course expected; the intuition is that TriAL queries have limited memory and thus cannot express queries such as the existence of a k-clique, for large values of k.

Expressivity of $\mathsf{TriAL}^=$. The TriAL queries we used to separate it from FO^5 or FO^4 make use of inequalities in the join conditions. Thus, it is natural to ask what happens when we restrict our attention to $\mathsf{TriAL}^=$, the fragment that disallows inequalities in selections and joins. We saw in Section 5 that this fragment appears to be more manageable in terms of query answering. This suggests that fewer variables may be enough, as the number of variables is often indicative of the complexity of query evaluation [23, 36]. This is indeed the case.

Theorem 5.

- FO^3 *is strictly contained in* $\mathsf{TriAL}^=$.
- $\mathsf{TriAL}^=$ *is strictly contained in* FO^4.

Next, we turn to the expressive power of TriAL^*. Since the Kleene star essentially defines the transitive closure of join operators, it seems natural for our study to compare TriAL^* with Transitive Closure Logic, or TrCl.

Formally, TrCl is defined as an extension of FO with the following operator. If $\varphi(\bar{x}, \bar{y}, \bar{z})$ is a formula, where $|\bar{x}| = |\bar{y}| = n$, and $\bar{u}, \bar{v}$ are tuples of variables of the same length n, then $[\mathbf{trcl}_{\bar{x},\bar{y}}\varphi(\bar{x}, \bar{y}, \bar{z})](\bar{u}, \bar{v})$ is a formula whose free variables are those in $\bar{z}$, $\bar{u}$ and $\bar{v}$. The semantics is as follows. For an instance I and an assignment $\bar{c}$ for variables $\bar{z}$, construct a graph G whose nodes are elements of I^n and edges contain pairs $(\bar{u}_1, \bar{u}_2)$ so that $\varphi(\bar{u}_1, \bar{u}_2, \bar{c})$ holds in I. Then $I \models [\mathbf{trcl}_{\bar{x},\bar{y}}\varphi(\bar{x}, \bar{y}, \bar{c})](\bar{a}, \bar{b})$ iff $(\bar{a}, \bar{b})$ is in the transitive closure of this graph G.

It is fairly easy to show that TriAL^* is contained in TrCl; the question is whether one can find analogs of Theorem 4 for fragments of TrCl using a limited number of variables. We denote by TrCl^k the restriction of TrCl to k variables. Note that constructs of form $[\mathbf{trcl}_{\bar{x},\bar{y}}\varphi(\bar{x}, \bar{y}, \bar{z})](\bar{t}_1, \bar{t}_2)$ can be defined using $|\bar{t}_1| + |\bar{t}_2| + |\bar{z}|$ variables, by reusing $\bar{t}_1$ and $\bar{t}_2$ in φ.

Then we can show that the relationship between TriAL^* and TrCl mimics the results of Theorem 4 for the case of TriAL and FO.

Theorem 6.

- TriAL^* *is strictly contained in* TrCl^6.
- TrCl^3 *is strictly contained in* TriAL^*.
- TriAL^* *is incomparable with* TrCl^4 *and* TrCl^5.

6.2 Triple Algebra as a Graph Language

The goal of this section is to demonstrate the usefulness of TriAL^* in the context of graph databases. In particular we show how to use TriAL^* for querying graph databases, both with and without data values, and compare it in terms of expressiveness with several well established graph database query languages.

6.2.1 Navigational graph query languages

We compare TriAL^* with a number of established formalisms for graph databases such as NREs, RPQs and *conjunctive* regular path queries (CRPQs). As our yardstick language for comparison we use a recently proposed version of XPath, adapted for graph querying [27]. Its navigational fragment, used now, is essentially Propositional Dynamic Logic (PDL) [21] with negation on paths; below we also expand it with data tests when we deal with graphs whose nodes hold data values. These languages are designed to query the topology of a graph database and specify various reachability patterns between nodes. As such, they are naturally equipped with the star operator and to make our comparison fair we will compare them with TriAL^*.

The navigational language used now is called GXPath; its formulae are split into node tests, returning sets of nodes and path expressions, returning sets of pairs of nodes.

Node tests are given by the following grammar:

$$\varphi, \psi := \top \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \langle\alpha\rangle$$

where α is a path expression.

The path formulae of GXPath are given below. Here a ranges over the labeling alphabet Σ.

$$\alpha, \beta := \varepsilon \mid a \mid a^- \mid [\varphi] \mid \alpha \cdot \beta \mid \alpha \cup \beta \mid \overline{\alpha} \mid \alpha^*.$$

The semantics is standard, and follows the usual semantics of PDL or XPath languages. Given a graph $G = (V, E)$, $\top$ returns V, and $\langle\alpha\rangle$ returns $v \in V$ so that (v, v') is in the semantics of α for some $v' \in V$. The semantics of Boolean operators is standard. For path formulae, ε returns $\{(v, v) \mid v \in V\}$, a returns $\{(v, v') \mid (v, a, v') \in E\}$ and a^- returns $\{(v', v) \mid (v, a, v') \in E\}$. Expressions $\alpha \cdot \beta, \alpha \cup \beta, \overline{\alpha}$, and α^* denote relation composition, union, complement, and transitive closure. Finally $[\varphi]$ denotes the set of pairs (v, v) so that v is in the semantics of φ.

Since TriAL^* is designed to query triplestores, we need to explain how to compare its power with that of graph query languages. Given a graph database $G = (V, E)$ over the alphabet Σ, we define a triplestore $T_G = (O, E)$, with $O = V \cup \Sigma$. Note that for now we deal with navigation; later we shall also look at data values.

To compare TriAL^* with binary graph queries in a graph query language $\mathcal{L}$, we turn TriAL^* ternary queries Q into binary by applying the $\pi_{1,3}(Q)$, i.e., keeping (s, o) from every triple (s, p, o) returned by Q. Under these conventions, we say that a graph query language $\mathcal{L}$ is contained in TriAL^* if for every binary query $\alpha \in \mathcal{L}$ there is a TriAL^* expression e_α so that $\pi_{1,3}(e_\alpha)$ and α are equivalent, and likewise, TriAL^* is contained in a graph query language $\mathcal{L}$ if for every expression e in TriAL^* there is a binary query $\alpha_e \in \mathcal{L}$ that is equivalent to $\pi_{1,3}(e)$. The notions of being strictly contained and incomparable extend in the same way.

Alternatively, one can do comparisons using triplestores represented as graph databases, as in Proposition 1. Since here we study the ability of TriAL^* to serve as a graph query language, the comparison explained above looks more natural, but in fact all the results remain true even if we do the comparison over triplestores represented as graph databases, as described in Section 2.

We now show that all GXPath queries can be defined in TriAL^*, but that there are certain properties that TriAL^* can define that lie beyond the reach of GXPath.

Theorem 7. *GXPath is strictly contained in* TriAL^*.

We prove this by using the equivalence of GXPath with the 3-variable fragment of reachability logic FO* [35], shown in [27].

Note that this also implies a strict containment of languages presented in [17, 18] in TriAL^*, since it is easy to show that they are subsumed by GXPath.

To compare TriAL^* with common graph languages such as NREs and RPQs we observe that NREs can be thought of as path expressions of GXPath that do not use complement and where nesting is replaced with $[\langle\alpha\rangle]$. RPQs do not even have nesting. Thus:

Corollary 2.

- *NREs are strictly contained in* TriAL^**.*
- *RPQs are strictly contained in* TriAL^**.*

It is common in graph databases to consider queries that are closed under conjunction and existential quantification, such as CRPQs [13, 37], C2RPQs [10] and CNREs [9]. The latter are expressions $\varphi(\bar{x}) = \exists \bar{y} \bigwedge_{i=1}^{n} (x_i \xrightarrow{e_i} y_i)$, where all variables x_i, y_i come from $\bar{x}, \bar{y}$ and each e_i is a NRE. The semantics extends that of NREs, with each $x_i \xrightarrow{e_i} y_i$ interpreted as the existence of a path between them that is denoted by e_i. We compare TriAL^* with these queries, and also with *unions* of CNREs that use bounded number of variables.

Theorem 8.

- *CNREs and* TriAL^* *are incomparable in terms of expressive power.*
- *Unions of CNREs that use only three variables are strictly contained in* TriAL^**.*

By observing that the expressions separating CNREs from TriAL^* are CRPQs, and that CNREs are more expressive than CRPQs and C2RPQS [8] we obtain:

Corollary 3.

- *CRPQs and* TriAL^* *are incomparable in terms of expressive power.*
- *Unions of C2RPQs and CRPQs that use only three variables are strictly contained in* TriAL^**.*

There are further extensions, such as *extended* CRPQs, where paths witnessing RPQs can be named and compared for relationships between them, defined as regular or even rational relations [6, 7]. We leave the comparison with these languages as future work.

6.2.2 Query languages for graphs with data

Until now we have compared our algebra with purely navigational formalisms. Triple stores do have data values, however, and can thus model any graph database. That is, for any graph database $G = (V, E, \rho)$ we can define a triplestore $T_G = (O, E, \rho)$ with $O = V \cup \Sigma$. Note that nodes corresponding to labels have no data values assigned in our model. This is not an obstacle and can in fact be used to model graph databases that have data values on both the nodes and the edges.

We provide a comparison to an extension of GXPath with data value comparisons. The language, denoted by $\mathsf{GXPath}(\sim)$, presented first in [27], is given by the following grammars for node and path formulae:

$$\varphi, \psi := \top | \langle \alpha = \beta \rangle \mid \langle \alpha \neq \beta \rangle \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \langle \alpha \rangle$$

$$\alpha, \beta := \varepsilon \mid a \mid a^- \mid [\varphi] \mid \alpha \cdot \beta \mid \alpha \cup \beta \mid \overline{\alpha} \mid \alpha^* \mid \alpha_= \mid \alpha_{\neq}.$$

The semantics of additional expressions is as follows: α_θ returns those pairs (v, v') returned by α for which $\rho(v)\ \theta\ \rho(v')$, for $\theta \in \{=, \neq\}$, and $\langle \alpha\ \theta\ \beta \rangle$ returns nodes v such that there are pairs (v, v_α) and (v, v_β) returned by α and β and $\rho(v_\alpha)\ \theta\ \rho(v_\beta)$. The former addition corresponds to the notion of regular expressions with equality [28], and the latter to standard XPath data-value comparisons.

To compare $\mathsf{GXPath}(\sim)$ with TriAL^*, we use the same convention as for data value-free languages. Connections of $\mathsf{GXPath}(\sim)$ with a 3-variable reachability logic and the proof of Theorem 4 show:

Corollary 4. *$\mathsf{GXPath}(\sim)$ is strictly contained in* TriAL^**.*

This also implies that TriAL^* subsumes an extension of RPQs based on regular expressions with equality [28], which can test for (in)equality of data values at the beginning and the end of paths.

Another formalism proposed for querying graph databases with data values is that of *register automata* [25]. In general, these work over data words, i.e., words over both a finite alphabet and an infinite set of data values. RPQs defined by register automata find pairs of nodes connected by a path accepted by such automata. We refer to [28, 25] for precise definitions, and state the comparison result below.

Proposition 6. TriAL^* *is incomparable in terms of expressive power with register automata.*

This follows since register automata can define properties not expressible with six variables, but on the other hand are not closed under complement.

7. CONCLUSIONS AND FUTURE WORK

While graph database query mechanisms have been promoted as a useful tool for querying RDF data, most of these approaches view RDF as a graph database. Although inherently similar, the two models do have significant differences. We showed that some very natural navigational queries for RDF cannot be expressed with graph-based navigational mechanisms. The solution is then to use proper triple-based models and languages.

We introduced such a model, that combines the usual idea of triplestores used in many RDF implementations, with that of graphs with data, and proposed an algebra for that model. It comes in two flavors, a non-recursive algebra TriAL and a recursive one TriAL^*. We also provided Datalog-based declarative languages capturing these. We studied the query evaluation problem,

as well as the expressivity of the languages, comparing them with both relational and graph query languages.

There are several future directions we would like to pursue. One relates to understanding connections with another way restriction guaranteeing closure, namely using semi-joins. Although some of the properties crucial for our goals cannot be expressed solely with semi-joins, such restrictions are closely related to the guarded fragment of FO [29], which enjoys better properties than the full FO. Another theoretical question that arises from our investigation is studying connections between languages for tuples of arbitrary arity, not just triples.

On the more practical side, we want to provide a deeper insight into the connection of our languages and nSPARQL, which seems to be the current choice for navigational RDF queries. For instance, we would like to see whether TriAL^* functionalities can be included into SPARQL, resulting in a language provably more expressive than nSPARQL, that provides recursive functionalities needed to compute most navigational queries required in RDF, including property paths. Another direction is to see how possible implementations of TriAL^* stack up against currently used systems. In this respect we would like to test if commercial RDBMSs can scalably implement the type of recursion we require, or whether augmenting one of the existing open-source triplestore systems will result in a more efficient evaluation when recursion is added.

Acknowledgments Work partly supported by EPSRC grants G049165 and J015377. We thank anonymous referees for their helpful suggestions.

8. References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] R. Angles and C. Gutierrez. Survey of graph database models. *ACM Computing Surveys*, 40(1), 2008.

[3] R. Angles. A comparison of current graph database models. In *ICDE Workshops*, pages 171–177, 2012.

[4] K. Anyanwu and A. Sheth. ρ-Queries: Enabling querying for semantic associations on the Semantic Web. In *WWW'03*, pages 690–699.

[5] M. Arenas and J. Pérez. Querying semantic web data with SPARQL. In *PODS*, pages 305–316, 2011.

[6] P. Barceló, L. Libkin, A.W. Lin, and P. Wood. Expressive languages for path queries over graph-structured data. *ACM TODS* 38(4) (2012).

[7] P. Barceló, D. Figueira, and L. Libkin. Graph logics with rational relations and the generalized intersection problem. In *LICS'12*, pages 115–124.

[8] P. Barceló, J. Pérez, and J. L. Reutter. Relative expressiveness of nested regular expressions. In *AMW'12*, pages 180–195.

[9] P. Barceló, J. Pérez, and J. L. Reutter. Schema mappings and data exchange for graph databases. In *ICDT'13*.

[10] D. Calvanese, G. De Giacomo, M. Lenzerini, and M.Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR'2000*, pages 176–185.

[11] D. Calvanese, G. De Giacomo, M. Lenzerini, and M.Y. Vardi. Rewriting of regular expressions and regular path queries. *JCSS*, 64(3):443–465, 2002.

[12] M. Consens, A. Mendelzon. GraphLog: a visual formalism for real life recursion. In *PODS'90*, pages 404–416.

[13] I. Cruz, A.O. Mendelzon, and P. Wood. A graphical query language supporting recursion. In *SIGMOD'87*, pages 323–330.

[14] P. Cudré-Mauroux and S. Elnikety. Graph data management systems for new application domains. *PVLDB*, 4(12):1510–1511, 2011.

[15] W. Fan, J. Li, S. Ma, N. Tang, and Y. Wu. Adding regular expressions to graph reachability and pattern queries. In *ICDE*, pages 39–50, 2011.

[16] W. Fan, J. Li, S. Ma, N. Tang, and Y. Wu. Graph pattern matching: from intractable to polynomial time. *PVLDB*, 3(1):264–275, 2010.

[17] G. Fletcher et al. Relative expressive power of navigational querying on graphs. *ICDT 2011*, 197-207

[18] G. Fletcher et al. The impact of transitive closure on the boolean expressiveness of navigational query languages on graphs. *FoIKS 2012*, 124-143

[19] G. Gottlob and C. Koch. Monadic datalog and the expressive power of languages for web information extraction. *J. ACM*, 51(1):74–113, 2004.

[20] G. Gottlob, E. Grädel, and H. Veith. Datalog LITE: a deductive query language with linear time model checking. *ACM TOCL*, 3(1):42–79, 2002.

[21] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.

[22] S. Harris et al. *SPARQL 1.1 Query Language*. `http://www.w3.org/TR/sparql11-query`.

[23] N. Immerman, D. Kozen. Definability with Bounded Number of Bound Variables. *IANDC*, 83(2):121-139 (1989).

[24] The Apache Jena Manual. `http://jena.apache.org`.

[25] M. Kaminski and N. Francez. Finite memory automata. *TCS*, 134(2):329–363, 1994.

[26] L. Libkin. *Elements of Finite Model Theory*, Springer, 2004.

[27] L. Libkin, W. Martens, and D. Vrgoč. Querying graph databases with XPath. In *ICDT*, 2013.

[28] L. Libkin and D. Vrgoč. Regular path queries on graphs with data. In *ICDT'12*, pages 74–85.

[29] D. Leinders, M. Marx, J. Tyszkiewicz and J. Van den Bussche. The semijoin algebra and the guarded fragment. *Logic, Language and Information*, 14(3), 331–343, 2009.

[30] K. Losemann, W. Martens. The complexity of evaluating path expressions in SPARQL. In *PODS'12*, pages 101–112.

[31] The Neo4j Manual. `http://docs.neo4j.org`.

[32] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM TODS*, 34(3), 2009.

[33] J. Pérez, M. Arenas, C. Gutierrez. nSPARQL: A navigational language for RDF. *J. Web Sem.*, 8(4):255–270, 2010.

[34] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C Recommendation 15 January 2008, `http://www.w3.org/TR/rdf-sparql-query/`.

[35] B. ten Cate. The expressivity of XPath with transitive closure. In *PODS*, pages 328–337, 2006.

[36] M. Vardi. On the complexity of bounded-variable queries. In *PODS'95*, pages 266–276.

[37] P. Wood. Query languages for graph databases. *Sigmod Record*, 41(1):50–60, 2012.

APPENDIX

Proofs

Remark 1: Throughout the appendix we will often denote conditions θ and η as conjunction of equalities or inequalities instead of sets. For example we will write $\theta = (1 \neq 3') \wedge (2 = 2')$ for $\theta = \{1 \neq 3', 2 = 2'\}$.

Remark 2: In the proofs we will usually handle only the case of the right Kleene closure $(R \bowtie)^*$. The proofs for the left closure are completely symmetric.

Remark 3: As usual in database theory, we only consider queries that are domain-independent, and therefore we loose no generality in assuming active domain semantics for FO formulas and other similar formalisms.

Proof of Proposition 1

Consider the RDF documents D_1 and D_2 consisting of the following triples:

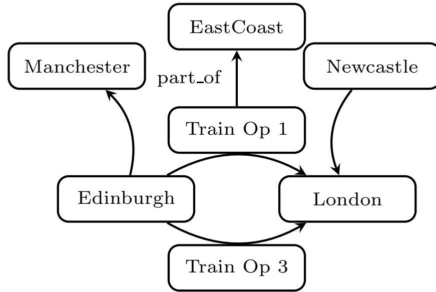
Graph D_1:

St Andrews	Bus Operator 1	Edinburgh
Edinburgh	Train Op 1	London
Edinburgh	Train Op 3	London
Edinburgh	Train Op 1	Manchester
Newcastle	Train Op 1	London
London	Train Op 2	Brussels
Bus Operator 1	part of	NatExpress
Train Op 1	part of	EastCoast
Train Op 2	part of	Eurostar
EastCoast	part of	NatExpress

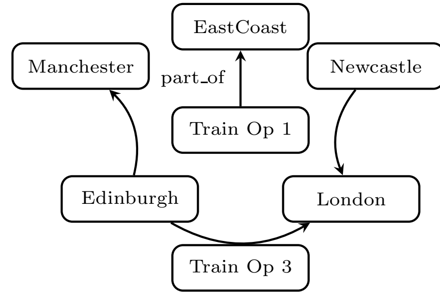
Graph D_2:

St Andrews	Bus Operator 1	Edinburgh
Edinburgh	Train Op 3	London
Edinburgh	Train Op 1	Manchester
Newcastle	Train Op 1	London
London	Train Op 2	Brussels
Bus Operator 1	part of	NatExpress
Train Op 1	part of	EastCoast
Train Op 2	part of	Eurostar
EastCoast	part of	NatExpress

Essentially, graph D_1 is an extension of the RDF document D in Figure 1, while graph D_2 is the same as D_1 except that it does not contain the triple (`Edinburgh`, `Train Op 1` , `London`). The relevant parts of our databases are illustrated in the following image.



Part of RDF graph D_1



Part of RDF graph D_2

The absence of this triple has severe implications with respect to the query Q of the statement of the Proposition, since in particular the pair (`St Andrews`, `London`) belongs to the evaluation of Q over D_1, but it does not belong to the evaluation of Q over D_2.

However, it is not difficult to check that the graph translations of D_1 and D_2 are exactly the same graph database: $\sigma(D_1) = \sigma(D_2)$. We have included the relevant part of transformations $\sigma(D_1)$ and $\sigma(D_2)$ in Figure **??**. It follows that Q is not expressible in nested regular expressions, since obviously the answer of all nested regular expressions is the same over $\sigma(D_1)$ and $\sigma(D_2)$ (they are the same graph).

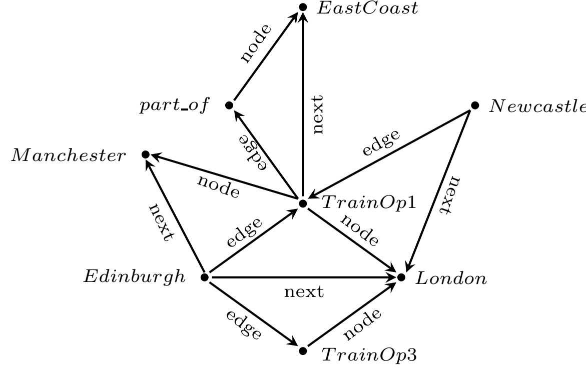


Figure 3: Transforming part of the RDF databases D_1 and D_2

Proof of Theorem 1

The semantics of the nested regular expressions in the RDF context (in [33]) is given as follows, assuming a triple representation of RDF documents. For next, it is the set $\{(v, v') \mid \exists z E(v, z, v')\}$, the semantics of edge is $\{(v, v') \mid \exists z E(v, v', z)\}$ and node is $\{(v, v') \mid \exists z E(z, v, v')\}$; for the rest of the operators it is the same as in the graph database case. Thus, even though stated in an RDF context, this semantics is essentially given according to the translation $\sigma(\cdot)$, in the sense that the semantics of an NRE e is the same for all RDF documents D and D' such that $\sigma(D) = \sigma(D')$ [1]. Hence the proof follows directly from Proposition 1 and the easy fact that Q cannot be expressed in SPARQL.

Proof of Proposition 2

Let us first show the containment of TriAL in non-recursive TripleDatalog$^\neg$. We show that for every expression e one can construct a non-recursive TripleDatalog$^\neg$ program Π_e such that, $e(T) = \Pi_e(I_T)$, for all triplestore databases T.

We define the translation by the following inductive construction, assuming Ans, Ans_1 and Ans_2 are special symbols that define the output of non-recursive TripleDatalog$^\neg$ programs.

- If e is just a triplestore name E, then Π_e consists of the single rule $Ans(x, y, z) \leftarrow E(x, y, z)$.
- If e is $e_1 \cup e_2$, then Π_e consists of the union of the rules of the programs Π_{e_1} and Π_{e_2}, together with the rules $Ans(\bar{x}) \leftarrow Ans_1(\bar{x})$ and $Ans(\bar{x}) \leftarrow Ans_2(\bar{x})$, where we assume that Ans_1 and Ans_2 are the predicates that define the output of Π_e and Π_e, respectively.
- If e is $e_1 - e_2$, then Π_e consists of the union of the rules of the programs Π_{e_1} and Π_{e_2}, together with the rule $Ans(\bar{x}) \leftarrow Ans_1(\bar{x}), \neg Ans_2(\bar{x})$, where we assume that Ans_1 and Ans_2 are the predicates that define the output of Π_{e_1} and Π_{e_2}, respectively.
- If e is $e_1 \bowtie^{i,j,k}_{\theta,\eta} e_2$, assume that θ consists of m conditions, and η consists of n conditions. Then Π_e consists of the union of the rules of the programs Π_{e_1} and Π_{e_2}, together with the rule

$$Ans(x_i, x_j, x_k) \leftarrow Ans_1(x_1, x_2, x_3), Ans_2(x_4, x_5, x_6), V(y_1, z_1), \ldots, V(y_n, z_n), u_1(=) \neq v_1, \ldots, u_m(=) \neq v_m,$$

  where for each p-th condition in θ of form $a = b$ or $a \neq b$, we have that $u_p = x_a$ and $v_p = x_b$ (or $u_p = o$ if a is an object o in $\mathcal{O}$, and likewise for b), and for each p-th condition in θ of form $\rho(a) = \rho(b)$ or $\rho(a) \neq \rho(b)$, we have that $y_p = x_a$ and $z_p = x_b$, and V is either $\sim$ or $\neg\sim$; and where we assume that Ans_1 and Ans_2 are the predicates that define the output of Π_{e_1} and Π_{e_2}, respectively.
- The case of selection goes along the same lines as the join case.

Clearly, this program is nonrecursive. Moreover, it is trivial to prove that this transition satisfies our desired property.

[1]The NRE's defined in [33] had additional primitives, such as next :: sp. These were added for the purpose of allowing RDFS inference with NREs, but play no role in the general expressivity of nSPARQL in our setting since we are dealing with arbitrary objects, whereas the constructs in [33] are limited to RDFS predicates.

Next we show the containment of non-recursive $\mathsf{TripleDatalog}^\neg$ in TriAL. We show that for every non-recursive $\mathsf{TripleDatalog}^\neg$ program Π one can construct an expression e_Π such that, $e_\Pi(T) = \Pi(I_T)$, for all triplestore databases T.

We assume that Π contains a single predicate Ans that represents the answer of the query. Also, without loss of generality we can assume that no rule uses predicate E, for some triplestore name E, other than a rule of form $P(x, y, z) \leftarrow E(x, y, z)$, for a predicate P in the predicates of Π that does not appear in the head of any other rule in Π.

We need some notation. The dependence graph of Π is a directed graph whose nodes are the predicates of π, and the edges capture the dependence relation of the predicates of Π, i.e., there is an edge from predicate R to predicate S if there is a rule in Π with R in its head and S in its body. Since Π is non-recursive, its dependency graph is acyclic. We now define the TriAL expression in a recursive fashion, following its dependency graph:

- Assume that all the rules in Π that have predicate S in the head are of form

  $$S(x_{a^j}, x_{b^j}, x_{c^j}) \leftarrow S_1^j(x_1^j, x_2^j, x_3^j), S_2^j(x_4^j, x_5^j, x_6^j), (\neg)\!\sim\!(y_1^j, z_1^j), \ldots, (\neg)\!\sim\!(y_n^j, z_n^j), u_1^j(\neq) = v_1^j, \ldots, u_m^j(\neq) = v_m^j$$

  for $1 \leq j \leq m$, and where S_1^j and S_2^j are (not necessarily distinct) predicate symbols of arity at most 3 and all variables in $x_{a^j}, x_{b^j}, x_{c^j}$ and each of y_i^j, z_i^j and u_k^j, v_k^j are contained in $\{x_1^j, x_2^j, x_3^j, x_4^j, x_5^j, x_6^j\}$.
  Then the TriAL expression e_S is

  $$\bigcup_{1 \leq j \leq m} e_{S_1^j} \bowtie_{\theta^j, \eta^j}^{a^j, b^j, c^j} e_{S_2^j},$$

  where θ contains an (in)equality $a = b$ for each (in)equality $x_a = x_b$ in the rule, and η^j contains an (in)equality $\rho(a) = \rho(b)$ for each predicate $\sim\!(a, b)$ (or its negation) in the rule. If either of S_1^j or S_2^j appear negated in the rule, then just replace $e_{S_1^j}$ for $(e_{S_1^j})^c$ or $(e_{S_2^j})^c$.
- The TriAL expression e_P (for predicate P in rule $P(x, y, z) \leftarrow E(x, y, z)$) is just E; if these variables appear in different order in the rule, we permute them via the selection operator σ.

It is now straightforward to verify that for every non-recursive $\mathsf{TripleDatalog}^\neg$ program Π whose answer predicate is Ans the expression e_{Ans} is such that, $e_{Ans}(T) = \Pi(I_T)$, for all triplestore databases T.

Proof of Theorem 2

Let us first show the containment of TriAL^* in $\mathsf{ReachTripleDatalog}^\neg$. The proof goes along the same lines as the proof of containment of TriAL in $\mathsf{TripleDatalog}^\neg$. We have to show that for every TriAL^* expression e there is a $\mathsf{ReachTripleDatalog}^\neg$ program Π_e such that $e(T) = \Pi_e(I_T)$, for all triplestores T.

The only difference from the construction in the proof of TriAL in $\mathsf{TripleDatalog}^\neg$ is the treatment of the constructs $e = (e_1 \bowtie_{\theta,\eta}^{i,j,k})^*$ and $e = (\bowtie_{\theta,\eta}^{i,j,k} e_1)^*$. For the former construct (the other one is symmetrical), assume that $\theta = (\bigwedge_{1 \leq i \leq m} p_i(\neq) = q_i)$ and $\eta = (\bigwedge_{1 \leq j \leq n} \rho(u_j)(\neq) = \rho(v_j))$. We let Π_e be the union of all rules of Π_{e_1}, plus rules

$$\begin{aligned} Ans(x, y, z) &\leftarrow Ans_1(x, y, z) \\ Ans(x_i, x_j, x_k) &\leftarrow Ans(x_1, x_2, x_3), Ans_1(x_4, x_5, x_6), \\ &\qquad (\neg)\!\sim\!(x_{p_1}, x_{q_1}), \ldots, (\neg)\!\sim\!(x_{u_n}, x_{v_n}), x_{p_1}(\neq) = x_{q_1}, \ldots, x_{p_m}(\neq) = x_{q_m}, \end{aligned}$$

where Ans_1 is the answer predicate of Π_{e_1}. Notice that we have assumed for simplicity there are no comparison with constants; these can be included in our translation the straightforward way. The proof that $e(T) = \Pi_e(I_T)$, for all triplestores T now follows easily.

The proof of containment of $\mathsf{ReachTripleDatalog}^\neg$ in TriAL^* also goes along the same lines as the proof that $\mathsf{TripleDatalog}^\neg$ is contained in TriAL. The only difference is when creating expression e_S, for some recursive predicate S. From the properties of $\mathsf{ReachTripleDatalog}^\neg$ programs, we know S is the head of exactly two rules of form

$$\begin{aligned} S(\bar{x}) &\leftarrow R(\bar{x}) \\ S(x_a, x_b, x_c) &\leftarrow S(x_1, x_2, x_3), R(x_4, x_5, x_6), V(y_1, z_1), \ldots, V(y_n, z_n), u_1(\neq) = v_1, \ldots, u_m(\neq) = v_m, \end{aligned}$$

1. R is a nonrecursive predicate of arity at most 3,

2. variables x_a, x_b, x_c and each of y_i, z_i and u_j, v_j are contained in $\{x_1, \dots, x_6\}$, and
3. each $V(y_i, z_i)$ is either $\sim(y_i, z_i)$ or $\neg\sim(y_i, z_i)$

We then let e_S be $(e_R \bowtie^{a,b,c}_{\theta,\eta})^*$, where θ contains the inequality $p(\neq) = q$ for each predicate $x_p(\neq) = x_q$ in the rule above, or the respective comparison with constant if p or q belong to $\mathcal{O}$, and η contains the (in)equality $\rho(p)(\neq) = \rho(q)$ for each predicate $\sim(x_p, x_q)$ (respectively, $\neg\sim(x_p, x_q)$).

Once again, it is straightforward to verify that e_{Ans} is such that, $e_{Ans}(T) = \Pi(I_T)$, for all triplestores T.

Proof of Proposition 3

The PTIME upper bound follows immediately from Theorem 3 below. PTIME-hardness follows from the fact that every TriAL query can be expressed in FO^6 (see Section 6) and the known result that evaluating FO^k queries is PTIME-hard already when $k = 3$ [36].

For the NLOGSPACE upper bound, the idea is to divide the expression e into all its subexpression, corresponding to subtrees of the parsing tree of φ. Starting from the leaves until the root of the parse tree of e, one can guess the relevant triples that will be witnessing the presence of the query triple in the answer set $e(T)$.

Note that for this we only need to remember $O(|e|)$ triples – a number of fixed length. After we have guessed a triple for each node in the parse tree for e we simply check that they belong to the result of applying the subexpression defined by that node of the tree to our triplestore T. Thus to check that the desired complexity bound holds we need to show that each of the operations can be performed in NLOGSPACE, given any of the triples. This follows by an easy inductive argument.

For example, if $e = E_i$ is one of the initial relations in T, we simply check that the guessed triple is present in its table. Note that this can be done in NLOGSPACE.

This is done in an analogous way for the expressions of the form $e = e_1 \cup e_2$ and $e = e_1 - e_2$. To see that the claim also holds for joins, note that one only has to check that join conditions can be verified in NLOGSPACE. But this is a straightforward consequence of the observation that for conditions we use only comparisons of objects and their data values.

Finally, to see that the star operator $(R \bowtie^{i,j,k}_{\theta,\eta})^*$ can be implemented in NLOGSPACE we simply do a standard reachability argument for graphs. That is, since we are trying to verify that a specific triple (a, b, c) is in the answer to the star-join operator, we guess the sequence that verifies this. We begin by a single triple in R (and we can check that it is there in NLOGSPACE by the induction hypothesis) and guess each new triple in R, joining it with the previous one, until we have performed at most $|T|$ steps.

Proof of Theorem 3

The basic outline of the algorithm is as follows:

1. Build the parse tree for our expression.
2. Evaluate the subexpressions bottom-up.

Now to see that the algorithm meets the desired time bounds we simply have to show that each step of evaluating a subexpression can be performed in time $O(|T|^2)$.

We prove this inductively on the structure of subexpression e.

As stated previously, we assume that the objects are sorted and that the triplestore is given by its adjacency matrix T with the property that $T[i, j, k] = 1$ if and only if $(o_i, o_j, o_k) \in T$. If we are dealing with a triplestore that has more than one relation we will assume that we assume to have access to each of the $n \times n \times n$ matrices representing E_i. In addition, to store data values we will use another array DV of size $|O|$ having $DV[i] = \rho(o_i)$, for $i = 1 \dots n$. In the end, our algorithm computes, given an expression e and a triplestore T the matrix R_e such that $(o_i, o_j, o_k) \in e(T)$ iff $R_e[i, j, k] = 1$.

If $e = E_i$, the name of one of the initial triplestore matrices, we already have our answer, so no computation is needed.

If $e = R_1 \cup R_2$ and we are given the matrix representation of R_1 and R_2 (that is the adjacency matrix of the answer of R_i on our triplestore T) we simply compute R_e as the union of these two matrices. Note that this takes time $O(|T|)$.

If $e = R_1 \cap R_2$ we compute R_e as the intersection of these two matrices. That is, for each triple (i, j, k) we check if $R_1[i, j, k] = R_2[i, j, k] = 1$. Note that this takes time $O(|T|)$.

If $e = R_1 - R_2$ we compute R_e as the difference of the two matrices. That is for each (i, j, k) we set $R_e[i, j, k] = 1$ if and only if $R_1[i, j, k] = 1$ and $R_2[i, j, k] = 0$. The time required is $O(|T|)$.

If $e = \sigma_\varphi R_1$ and we are given the matrix for R_1 we can compute R_e in time $O(|e||T|)$ by traversing each triple (i, j, k), checking that $R_1[i, j, k] = 1$ and that the objects o_i, o_j and o_k satisfy the conditions specified by φ. Notice that each of these checks can be done in $|e|$ time using T and DV, since the number of comparisons in φ has a fixed upper bound, modulo comparison with constants. The comparison with constants can be done in time $|e|$ because we have to check (in)equality only with the constants that actually appear in e.

Finally, in the case that $e = R_1 \bowtie_{\theta,\eta}^{i',j',k'} R_2$ we can compute R_e using the following algorithm:

: Computing joins

[1] Matrix representation of R_1, R_2 Matrix R_e representing e Let θ' and η' be the conditions obtained from θ, η by removing comparisons with constants Let α, β be the conditions in θ, η using constants Filter R_1 and R_2 according to α, β $i = 1 \to n$ $j = 1 \to n$ $k = 1 \to n$ $R_1[i, j, k] = 1$ $l = 1 \to n$ $m = 1 \to n$ $n = 1 \to n$ $R_2[l, m, n] = 1$ (o_i, o_j, o_k) and (o_l, o_m, o_n) satisfy the conditions in θ', η' $R_e[i', j', k'] = 1$ $R_e[i', j', k'] = 0$

Note that lines 1–3 correspond to computing selections operator and can therefore be performed using the time $O(|e||T|)$ and reusing the matrices R_1 and R_2. It is straightforward to see that the remaining of the algorithm works as intended by joining the desirable triples. This is performed in $O(|T|^2)$. Thus the whole join computation can be done in time $O(|T|^2)$.
This concludes the first part of our theorem and we thus conclude that TriAL query computation problem can be solved in time $O(|e||T|^2)$.
For the second part of the theorem we only have to show that each star operation can be computed in time $O(|T|^3)$. To see this we consider the following algorithm, computing the answer set for $e = (R_1 \bowtie_{\theta,\eta}^{i',j',k'})^*$

: Computing stars

[1] Matrix representation of R_1 Matrix R_e representing e Initialize $R_e := R_1$ $i = 1 \to n^3$ Compute $R_e := R_e \cup R_e \bowtie_{\theta,\eta}^{i',j',k'} R_1$

First we note that the algorithm does indeed compute the correct answer set. This follows because the joining in our star process has to became saturated after n^3 steps, since this is the maximum possible number of triples in a model with n elements. Note now that each join in step 3 can be computed in time $O(|T|^2)$, thus giving us the total running time of $O(n^3 \cdot |T|^2) = O(|T|^3)$.
Finally, note that left-joins can be computed in an analogous way.

Proof of Proposition 4

To prove this we will use the close connection of positive fragment of TriAL$^=$ with FO^4. We establish this as follows. To each triplestore $T = (O, E_1, \ldots, E_n, \rho)$ we associate an FO structure $\mathcal{M}_T = (O, E_1, \ldots, E_n, \sim)$, where O is the set of objects appearing in T, $E_1, \ldots, E_n$ are just the representation of the triplestores, and $\sim(o_1, o_2)$ holds iff $\rho(o_1) = \rho(o_2)$ (they have the same data value).

Then we can show

Lemma 1. *For every TriAL$^=$ expression e one can construct an FO^4 formula φ_e such that a triple (a, b, c) belongs to $e(T)$ if and only if $\mathcal{M}_T \models \varphi_e(a, b, c)$.*

Proof. The proof is done by induction. The base case when $e = E_i$ for some $1 \le i \le n$ is trivial, and so are the cases when $e = e_1 \cup e_2$, $e = e_1 - e_2$ and $e = \sigma_{\theta,\eta} e_1$. The only interesting case is when $e = e_1 \bowtie_{\theta,\eta}^{i,j,k} e_2$.

As usual, we assume that e is $e_1 \bowtie_{\theta,\eta}^{i,j,k} e_2$, where θ is a conjunction of equalities between elements in $\{1, 1', 2, 2', 3, 3'\} \cup \mathcal{O}$ and η is a conjunction of equalities between elements in $\{\rho(1), \rho(1').\rho(2), \rho(2'), \rho(3), \rho(3')\}$. We need some terminology.

Let $\theta = \theta_\ell \wedge \theta_r \wedge \theta_\bowtie \wedge \theta_\ell^c \wedge \theta_r^c$, where

- θ_ℓ and θ_r contain only equalities between indexes in $\{1, 2, 3\}$ and $\{1', 2', 3'\}$, respectively.

- θ^c_ℓ and θ^c_r contain only equalities where one element is in $\mathcal{O}$ and the other is in $\{1,2,3\}$ and $\{1',2',3'\}$, respectively.
- $\theta_\bowtie$ contains all the remaining equalities, i.e. those equalities in which one index is in $\{1,2,3\}$ and the other in $\{1',2',3'\}$.

We also divide $\eta = \eta_\ell \wedge \eta_r \wedge \eta_\bowtie$ in the same fashion (recall that for the sake of readability we assume no comparison between data values and constants, two avoid two sorted structures). Notice that any two equalities of form $i = j'$ and $i = k'$, for $i \in \{1,2,3\}$ and $j', k' \in \{1',2',3'\}$ can be replaced with $i = j'$ and $j' = k'$, and likewise we can replace $i = k'$ and $j = k'$ with $i = j$ and $j = k'$. For this reason we assume that $\theta_\bowtie$ (and $\eta_\bowtie$) contain at most 3 inequalities, and no two inequalities in them can mention the same element. Furthermore, if $\theta_\bowtie$ has two or more equalities, then the join can be straightforwardly expressed in FO^4, since now instead of the six possible positions we only care about four -or three-of them. For this reason we only show how to construct the formula when $\theta_\bowtie$ has one or no equalities.

Finally, for a conjunction θ of equalities between element in $\{1,1',2,2',3,3'\}$, we let $\alpha(\theta)$ be the formula $\bigwedge_{i=j\in\theta} x_i = x_j$, for a conjunction η of equalities of elements in $\{\rho(1),\rho(1').\rho(2),\rho(2'),\rho(3),\rho(3')\}$, let $\beta(\eta)$ be the formula $\bigwedge_{\rho(i)=\rho(j)\in\eta} \sim(x_i,x_j)$, and for a conjunction θ^c of equalities between an object in $\mathcal{O}$ and an element in $\{1,1',2,2',3,3'\}$ we let $\alpha(\theta^c) = \bigwedge_{o=i\in\theta^c} o = x_i$.

In order to construct formula φ_e, we distinguish 2 types of joins:

- Joins of form $e = e_1 \bowtie^{i,j,k}_{\theta,\eta} e_2$ where all of i,j,k belong to either $\{1,2,3\}$ or $\{1',2',3'\}$.

  Assume that i,j,k belong to $\{1,2,3\}$ (the other case is of course symmetrical). We first consider the case in which $\theta_\bowtie$ has no equalities, while $\eta_\bowtie$ has three equalities. Moreover, assume for the sake of readability that $\eta_\bowtie = (\rho(1) = \rho(1')) \wedge (\rho(2) = \rho(2')) \wedge (\rho(3) = \rho(3'))$. We then let

  $$\varphi_e(x_i,x_j,x_k) = \varphi_{e_1}(x_1,x_2,x_3) \wedge \alpha(\theta_\ell) \wedge \alpha(\theta^c_\ell) \wedge \beta(\eta_\ell) \wedge \exists w\Big(\sim(x_1,w) \wedge \exists x_1(\sim(x_2,x_1) \wedge \exists x_2(\sim(x_3,x_2)\varphi_{e_2}(w,x_1,x_2) \wedge \alpha(\theta_r)[x_{1'},x_{2'},x_{3'} \to w,x_1,x_2] \wedge \alpha(\theta^c_r)[x_{1'},x_{2'},x_{3'} \to w,x_1,x_2] \wedge \beta(\eta_r)[x_{1'},x_{2'},x_{3'} \to w,x_1,x_2]))\Big)$$

  Where a formula $\psi[x,y,z \to x',y',z']$ is just the formula ψ in which we replace each occurrence of variables x,y,z for x',y',z', respectively. For the case when $\theta_\bowtie$ is nonempty, notice here than any equality in $\theta_\bowtie$ only makes our life easier, since it eliminates one of the existential guesses we need in the above formula. Furthermore, if $\eta_\bowtie$ has less equalities, then we just remove the corresponding $\sim$ predicates. This cover all other possible cases of $\theta_\bowtie$ and $\eta_\bowtie$.

  Let us illustrate this construction with an example.

  Example 6. *Consider the expression $e = e_1 \bowtie^{1,2,3}_{1=2\wedge\rho(2)=\rho(2')\wedge\rho(2')=\rho(3')} e_2$. Then θ_ℓ is $1 = 2$, $\eta_\bowtie$ is $\rho(2) = \rho(2')$ and $\eta_r = \rho(2') = \rho(3')$, all of the remaining formulas being empty. Then we have:*

  $$\varphi_e(x_1,x_2,x_3) = \varphi_{e_1}(x_1,x_2,x_3) \wedge x_1 = x_2 \wedge \exists w\Big(\exists x_1(\sim(x_1,x_2) \wedge \exists x_2(\varphi_{e_2}(w,x_1,x_2) \wedge \sim(x_1,x_2)))\Big)$$

- Joins of form $e = e_1 \bowtie^{i,j,k}_{\theta,\eta} e_2$ where not all of i,j,k belong to either $\{1,2,3\}$ or $\{1',2',3'\}$. Assume for the sake of readability that $i = 1$, $j = 2$ and $k = 3'$ (all of other cases are completely symmetrical). We have again two possibilities.

  (-) There are no equalities in $\theta_\bowtie$. Assume that $\eta_\bowtie = (\rho(1) = \rho(1')) \wedge (\rho(2) = \rho(2')) \wedge (\rho(3) = \rho(3'))$ (we have already proved that there are at most 3 equalities in η'), cases with less equalities are treated along the same lines. We then let

  $$\varphi_e(x_1,x_2,x_{3'}) = \big(\exists x_3(\varphi_{e_1}(x_1,x_2,x_3) \wedge \alpha(\theta_\ell) \wedge \alpha(\theta^c_\ell) \wedge \beta(\eta_\ell)) \wedge \sim(x_3,x_{3'})\big) \wedge \exists x_3\Big(\sim(x_1,x_3) \wedge \exists x_1(\sim(x_2,x_1) \wedge \varphi_{e_2}(x_3,x_1,x_{3'}) \wedge \alpha(\theta_r)[x_{1'},x_{2'} \to x_3,x_1] \wedge \alpha(\theta^c_r)[x_{1'},x_{2'} \to x_3,x_1] \wedge \beta(\eta_r)[x_{1'},x_{2'} \to x_3,x_1]\Big)$$

(-) There is a single equality in $\theta_{\bowtie}$. Assume for the sake of readability that $i = 1$, $j = 2$ and $k = 3'$ (all of other cases are completely symmetrical). Notice that if $\theta_{\bowtie}$ has the equality $3 = 3'$, then this is equivalent to the previous case with one equality in $\theta_{\bowtie}$, but with $k = 3$. Moreover, equalities in $\theta_{\bowtie}$ involving 1 or 2 just make our life easier, so we will also not take them into account here. We are thus left with the assumption that $\theta_{\bowtie}$ contains the equality $3 = 1'$ (the case where it contains instead $3 = 2'$ is symmetrical)
Moreover, assume as well that $\eta_{\bowtie} = (\rho(1) = \rho(1')) \wedge (\rho(2) = \rho(2')) \wedge (\rho(3) = \rho(3'))$ (we have already proved that there are at most 3 equalities in $\eta_{\bowtie}$, and from the form of the formula it is clear that all other cases are treated along the same lines).
We then let

$$\varphi_e(x_1, x_2, x_{3'}) = \\ \exists x_{1'} \Big(\varphi_{e_1}(x_1, x_2, x_{1'}) \wedge \alpha(\theta_\ell)[x_3 \to x_{1'}] \wedge \alpha(\theta_\ell^c)[x_3 \to x_{1'}] \wedge \beta(\eta_\ell)[x_3 \to x_{1'}] \wedge \sim(x_1, x_{1'}) \wedge \\ \exists x_1 \big(\sim(x_1, x_2) \wedge \varphi_{e_2}(x_{1'}, x_1, x_{3'}) \wedge x_{1'} = x_{3'} \wedge \alpha(\theta_r)[x_{2'} \to x_1] \wedge \alpha(\theta_r^c)[x_{2'} \to x_1] \wedge \beta(\eta_r)[x_{2'} \to x_1]\big)\Big)$$

Having established how to construct φ_e, it is now straightforward to show that it satisfies the property of the Lemma. □

We now continue with the proof of the Proposition. Note that we can compute $\mathcal{M}_T$ from T in linear time. Additionally, observe that the size of $\mathcal{M}_T$, when defined as a size of its encoding (see [26]) is linear in T. Also, observe that the formula φ_e corresponding to e is of size $O(|e|)$.

To finish the proof we show that each FO^4 formula φ using relations that are at most ternary (in fact this holds for relations of arity four as well, but is not relevant for our analysis) can be evaluated in time $O(|F| \cdot |O|^4)$.

Lemma 2. *Let φ be an arbitrary formula using at most four variables. Then the set of all tuples that make φ true in $\mathcal{M}$, with $\mathcal{M}$ as above (we omit the subscript T for the sake of readability, since it is now clear), can be computed in time $O(|F| \cdot |O|^4)$.*

PROOF. To see that this holds note that we can assume that our formulas only use the connectives $\neg, \vee$ and the quantifier $\exists$. Indeed, we can assume this since any formula using other quantifiers can be rewritten using the ones above with a constant blow-up in the size of formula. In particular, our formulas in Lemma **??** use only $\wedge$ in addition to these three logical connectives, and $\wedge$ can be rewritten in terms of $\vee$ and $\neg$.

The desired algorithm works as follows.

1. Build a parse tree for the formula φ.
2. Compute the output relation(s) bottom-up using the tree.

To see that the algorithm works with the desired time bound we only have to make sure that each of the computation steps in 2 can be performed in time $O(|O|^4)$. We have three cases to consider, based on whether we are using negation, disjunction, or existential quantification. Here we assume that we compute a matrix $\psi(\mathcal{M})$, for each subformula ψ of φ. Note that, since we use formulas with at most four free variables each matrix can be of size at most $|O|^4$ (i.e. we are working with a four dimensional matrix). If the (sub)formula has only two free variables the resulting matrix will, of course, be two dimensional.

First we consider the case of negation. That is, assume that we have a matrix $\psi(\mathcal{M})$ and we are evaluating a formula $\varphi = \neg\psi$. Then we simply build a matrix for the $\varphi(\mathcal{M})$ by flipping each bit in the matrix for $\psi(\mathcal{M})$. This can clearly be done in time $O(|O|^4)$ by traversing the entire matrix.

Next, consider the case when $\varphi = \exists x \psi(x, y, z, w)$ and assume that we have the matrix for $\psi(x, y, z, w)$. The existing matrix is now reduced to a three dimensional matrix with the value 1 in position i, j, k if and only if there is an l such that $\psi(\mathcal{M})[l, i, j, k] = 1$. Note that computing this amounts to scanning the entire matrix for ψ. In the case when ψ case only three free variables we will need only $O(|O|^3)$ time to compute $\varphi(\mathcal{M})$.

Finally, let $\varphi = \psi_1(x, y, w) \vee \psi_2(x, y, z, w)$. The cases when ψ_1 and ψ_2 have a different number of free variables follows by symmetry. What we do first is to compute a 4-D matrix $\psi_1'(\mathcal{M})$ by setting $\psi_1'(\mathcal{M})[i, j, k, l] = 1$ iff $\psi_1(\mathcal{M})[i, j, l] = 1$. Note that this matrix can be computed in time $O(|O|^4)$. Next we compute the output matrix by putting 1 in each cell where either $\psi_1'(\mathcal{M})$ or $\psi_2(\mathcal{M})$ have 1. All the other cases can be performed symmetrically by using the appropriate matrices and their projections.

This completes the proof of our lemma. □

The result of Proposition 4 now follows, since we can take our expression e, transform it into a formula φ_e of FO^4 and evaluate it in time $O(|\varphi_e| \cdot |O|^4) = O(|e| \cdot |O| \cdot |T|)$, since $|T| = |O|^3$ and $|\varphi_e| = O(|e|)$.

Proof of Proposition 5

To show this we will use the algorithm presented in Proposition 4. All of the operations except the evaluation of Kleene star will be preformed in a same way as there. Note that we can assume this since the algorithm in Lemma **??** computes the subexpressions bottom up using the matrices representing the output. Thus we can use it to compute answers to subformulas, compose it with the method presented here to evaluate Kleene stars and proceed with the algorithm from Lemma **??**. To obtain the desired complexity bound we only have to show how to compute navigational operations in time $O(|O| \cdot |T|)$.

That is, we show how to, given a matrix representation for a relation R we compute matrix representation for $(R \bowtie^{1,2,3'}_{3=1'})^*$ and $(R \bowtie^{1,2,3'}_{3=1',2=2'})^*$, respectively.

Let $O = \{o_1, \ldots, o_n\}$ be the set of object appearing in our triplestore T. (The assumption that they are ordered is standard when considering matrix representations). As input, we are given a three dimensional matrix R representing the output of relation R when evaluated over T. That is we have $(o_i, o_j, o_k) \in R(T)$ if and only if $R[i, j, k] = 1$. (Here we use R both to denote the relation R and its matrix representation).

First we give a procedure that computes the matrix M_e for the expression

$$e = (R \bowtie^{1,2,3'}_{3=1'})^*.$$

: Computing $e = (R \bowtie^{1,2,3'}_{3=1'})^*$

[1] Matrix representation of R Matrix M_e representing e Precomputing the reachability matrix R_{reach}: $i = 1 \to n$ $j = 1 \to n$ $k = 1 \to n$ $R[i,k,j] = 1$ $R_{reach}[i,j] = 1$ Compute the transitive closure R^*_{reach} Compute the output matrix M_e: $i = 1 \to n$ $j = 1 \to n$ $k = 1 \to n$ $R[i,k,j] = 1$ $l = 1 \to n$ $R^*_{reach}[j,l] = 1$ $M_e[i,k,l] = 1$

To show that the algorithm works correctly notice that steps 1 to 6 precompute the matrix R_{reach} such that $R_{reach}[i, j] = 1$ if and only if o_i has and out edge ending in o_j (or equivalently $(o_i, o, o_k) \in T$ for some o). After this in step 7 we compute the transitive closure R^*_{reach} thus obtaining all pairs of nodes reachable one from another using path of arbitrary label in the graph representing T. Next in steps 8 to 15 we simply compute all the triples in the output matrix M_e. To do so we observe that a pair (o_i, o_k) will belong to some triple (o_i, o_k, o_l) of the output, if there is j such that $(o_i, o_k, o_j) \in T$ (line 12) and o_l is reachable from o_j (line 14).
To determine the complexity of the algorithm notice that steps 1 to 6 take time $O(|O|^3) = O(|T|)$, while computing the transitive closure in step 7 can be done using Warshall's algorithm (see T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, Introduction to Algorithms, The MIT Press, 2003.) in time $O(|O|^3) = O(|T|)$. Finally steps 8 to 15 take time $O(|O| \cdot |T|)$, thus giving us the desired time bound.
Next we show how to compute joins of the form $(R \bowtie^{1,2,3'}_{3=1',2=2'})^*$ using a slight modification of the algorithm above.

: Computing $e = (R \bowtie^{1,2,3'}_{3=1',2=2'})^*$

[1] Matrix representation of R Matrix M_e representing e $k = 1 \to n$ Precomputing the reachability matrix R^k_{reach}: $i = 1 \to n$ $j = 1 \to n$ $R[i,k,j] = 1$ $R_{reach}[i,j] = 1$ Compute the transitive closure ${R^k_{reach}}^*$ compute the output matrix M_e: $i = 1 \to n$ $j = 1 \to n$ $R[i,k,j] = 1$ $l = 1 \to n$ ${R^k_{reach}}^*[j,l] = 1$ $M_e[i,k,l] = 1$

It is straightforward to see that the algorithm uses the same time to compute the output as the algorithm in Procedure 1.
To show that it works correctly observe that we precompute matrix R^k_{reach} for each k, thus checking reachability only for triples whose second node is o_k. Since the rest of the algorithm works in the same way as the one in Procedure 1, we conclude that the computed answer M_e represents e correctly.
The case when the number of labels is fixed now follows by observing that the values for the loops defined by k in both algorithms come from a set of constant length.

Proof of Theorem 4

We begin with **Part 1**.

Let e be a TriAL expression. We construct an FO^6 formula φ_e such that $e(T) = \varphi_e(I_T)$, for each triplestore T. The proof is by induction.

- For the base case, if e corresponds to a triplestore name E, then φ_e is $E(x,y,z)$.
- If $e = e_1 \cup e_2$, then $\varphi_e(x,y,z) = \varphi_{e_1}(x,y,z) \vee \varphi_{e_2}(x,y,z)$, which clearly is in FO^6 since existential variables within φ_{e_1} and φ_{e_2} can be renamed and reused.
- If $e = e_1 - e_2$, then $\varphi_e(x,y,z) = \varphi_{e_1}(x,y,z) \wedge \neg\varphi_{e_2}(x,y,z)$
- If $e = e_1 \bowtie^{i,j,k}_{\theta,\eta} e_2$, then $\varphi_e(x_i,x_j,x_k) = \exists x_u \exists x_v \exists x_w \varphi_{e_1}(x_1,x_2,x_3) \wedge \varphi_{e_2}(x_{1'},x_{2'},x_{3'}) \wedge \alpha(\theta) \wedge \beta(\eta)$, where u,v,w are the remaining elements that together with i,j,k complete $\{1,1',2,2',3,3'\}$, $\alpha(\theta)$ contains the equality $x_p = x_q$ or $x_p = o$ for each equality $p = q$ or $p = o$ in θ, for $o \in \mathcal{O}$ and $p,q \in \{1,1',2,2',3,3'\}$, and likewise for inequalities, and $\beta(\eta)$ contains atom $\sim(x_p,x_q)$ for each equality $\rho(p) = \rho(q)$ in η, and likewise for inequalities using atom $\neg\sim$.
- Similarly, if $e = \sigma_{\theta,\eta} e_1$ then $\varphi_e(x,y,z) = \varphi_{e_1}(x,y,z) \wedge \alpha(\theta) \wedge \beta(\eta)$, where $\alpha(\theta)$ and $\beta(\eta)$ are defined as in the previous bullet.

It is now straightforward to check the desired properties for e and φ_e.

That the containment is strict follows from Part 3 of the proof.

Next we move onto **Part 2**

To show that FO^3 is contained in TriAL, one needs to show how to construct, for every FO^3 formula φ, an equivalent TriAL expression e_φ such that $e_\varphi(T) = \varphi(I_T)$, for all triplestores T.

The construction is done by induction on the formula.

Recall here that U is just a shorthand for the relation that contains O^3.

- For the base case, if $\varphi = E(x_1,x_2,x_3)$ for some triplestore name then e_φ is just E. However, in the general case when $\varphi = E(x_i,x_j,x_k)$, for each of i,j,k in $\{1,2,3\}$, we let $e_\varphi = E \bowtie^{i,j,k} E$. For the other base case, if φ is $x_1 = x_2$, then $e_\varphi = \sigma_{1=2} U$.
- If $\varphi = \neg\varphi_1$, then $e_\varphi = U - e_{\varphi_1}$ (recall that we assume active domain semantics for FO formula).
- If $\varphi = \exists x \varphi_1(\bar{y})$, then $e_\varphi = e_{\varphi_1} \bowtie^{\bar{d}} U$, where $\bar{d}$ depends on the size of $\bar{y}$: if $|\bar{y}| = 3$ then $\bar{d} = i,j,k'$, if $\bar{y} = 2$ then $\bar{d} = i,j',k'$, and if $\bar{y} = 1$ then $\bar{d} = i',j',k'$.
- If $\varphi = \varphi_1(\bar{x},\bar{y}) \vee \varphi_2(\bar{x},\bar{z})$, then $e_\varphi = e_{\varphi_1} \cup e_{\varphi_2}$. Notice here that we assume that variables in $\bar{x},\bar{y},\bar{z}$ appear in the same order in both φ_1 and φ_2. If this is not the case then one can only permute the variables by doing a join, as in the base case.

We leave the proof that φ and e_φ satisfy our desired properties, since it is easy to check. The key idea is that we do not need projection in our algebra to simulate FO^3 queries, since we know that they will have 3 free variables at the end, in the induction step we can just ignore some of the positions in the triples.

To show that the containment is proper, consider the following property over triplestore databases:

A triplestore database T has four different objects.

It is not difficult to construct a TriAL expression e such that $e(T)$ is nonempty if and only if T has four different objects. For example, one can use the expression $e = U \bowtie^{1,2,3}_{\theta} U$, where $\theta = (1 \neq 2) \wedge (1 \neq 3) \wedge (1 \neq 1') \wedge (2 \neq 3) \wedge (2 \neq 1') \wedge (3 \neq 1')$.

On the other hand, let $T_3 = (O_3, E_3, \rho)$ be the triplestore in which $O_3 = \{a,b,c\}$ and $E_3 = O_3 \times O_3 \times O_3$, and $T_4 = (O_4, E_4, \rho')$ be the triplestore in which $O_4 = \{a,b,c,d\}$ and $E_4 = O_4 \times O_4 \times O_4$. In addition we set $\rho(a) = \rho(b) = \rho(c) = 1$ and $\rho' = \rho \cup \{(d,1)\}$. It is trivial to show that these structures cannot be distinguished by any formula in the infinitary logic $\mathcal{L}^3_{\infty\omega}$ [26], since the duplicator always has a strategy to ensure that the 3-pebble game can be played forever in these structures (see e.g. [26]). Note that the standard game will work here, since all the data values are the same, so they do not influence the winning strategy of the duplicator. It follows that the expression e cannot be expressed in FO^3 (in fact, not even in $\mathcal{L}^3_{\infty 3}$).

For **Part 3**, we show that TriAL is incomparable with FO^4 and FO^5.

We begin by showing that the following TriAL query:

$$e_6 := U \overset{1,2,3}{\underset{\theta}{\bowtie}} U, \text{ with } \theta = \bigwedge_{i,j \in \{1,2,3,1',2',3'\}, i \neq j} i \neq j,$$

cannot be expressed in FO^5 (and thus not in FO^4).

Note that this is a modification of the query from part 1 of this proof that simply states that our triplestore has at least six objects. Now take $T_5 = (O_5, E_5, \rho)$ with $O_5 = \{a, b, c, d, e\}$, and $E_5 = O_5 \times O_5 \times O_5$, where ρ assigns the same data value to all elements of O_5 and define O_6 in an analogous way, but with six elements. It is a well known fact [26] that the duplicator has a winning strategy in a 5-pebble game on these two structures, so they can not be distinguished by an FO^5 formula. On the other hand our expression e_6 does distinguish them and is thus not expressible in FO^5.

Next we show that there is an FO^4 expression that cannot be expressed by any TriAL query (and thus TriAL cannot express neither full FO^5 nor FO^6. In order to do that, we first need to show that triple algebra expressions can be expressed with a particular extension of FO^3, that we call here FO^3-join.

Formally, we construct FO^3-join formulas from FO^3 formulas, the usual operators of disjunction, conjunction, negation, existential and universal quantification, and the following join operator: if φ_1 and φ_2 are formulas in FO^3-join that use variables x_1, x_2, x_3 and $x_{1'}, x_{2'}, x_{3'}$ respectively, θ is a conjunction of equalities between indexes in $\{1, 1', 2'2', 3, 3'\}$ and η is a conjunction of equalities between indexes in $\rho(1), \ldots, \rho(3')$, then the formula $\varphi(x_i, x_j, x_k) = \varphi_1(x_1, x_2, x_3) \bowtie^{i,j,k}_{\theta,\eta} \varphi_2(x_{1'}, x_{2'}, x_{3'})$ is a formula in FO^3-join that only uses variables x_i, x_j, x_k. Furthermore, the number of variables in FO^3-join formulas is restricted to 3, but note that for the sake of counting variables the construct $\varphi(x_i, x_j, x_k) = \varphi_1(x_1, x_2, x_3) \bowtie^{i,j,k}_{\theta,\eta} \varphi_2(x_{1'}, x_{2'}, x_{3'})$ is assumed to use only variables x_i, x_j and x_k.

The semantics of the join construct is defined in the same way than Triple Algebra, and the rest of the operators is defined in the same way as FO. It is now not difficult to show the following:

Lemma 3. *Triple Algebra is contained in* FO^3*-join.*

In fact, one can actually show that both languages have the same expressive power, but for the sake of this proof we will not bother. Continuing with the proof, we now define a game that characterizes expressibility in FO^3-join.

Let J be the set of all the join symbols that we allow in TriAL. A *recipe* p for FO^3-join is a tree of rank 2 (i.e., every node can have at most two children) labeled with symbols from alphabet $\{\exists, \forall\} \cup \mathcal{J}$, such that the following holds: If a node n of p has two children, then it labeled with a symbol in J, and if a node n of p has one children, then it is labeled with $\exists$ or $\forall$.

For every such recipe p, define the *quantifier class* $L(p)$ inductively as follows:

- $L(\varepsilon)$ contains quantifier and join free formulae.
- If the root of p is labeled with $Q \in \{\exists, \forall\}$, then $L(p)$ is the closure under conjunctions and disjunctions of the class $L(p') \cup \{Qx\varphi \mid \varphi \in L(p')\}$, where p' is the subtree of p whose root is the only child of p.
- If the root of p is labeled with a symbol $\bowtie$ in J, let p_1 and p_2 be the subtrees of p whose root are the first and second children of p, respectively. Then $L(p)$ is the closure under conjunctions and disjunctions of the class of all formulae $\varphi \bowtie \psi$, where $\varphi \in L(p_1)$ and $\psi \in L(p_2)$.

We now define the join game between two structures. This game proceeds as in a typical 3-pebble game (see [26] for a precise explanation), except the following sets of moves are available to the spoiler:

The join $\bowtie^{i,j,k}_{\theta,\eta}$ move:
The spoiler picks a structure, and then splits the 3 pebbles in that structure into two sets of 3 pebbles, set 1 and set 2, with the condition that the split *satisfies* the join: If before the move the first, second and third pebbles where in elements a, b and c, then the first, second and third elements of each of the set of pebbles must be placed in elements a_1, b_2, c_1 and a_2, b_2, c_2 such that $(a, b, c) = (a_1, b_1, c_1) \bowtie^{i,j,k}_{\theta,\eta} (a_2, b_2, c_2)$.

Duplicator must then split the pebbles in the other structure into two sets of pebbles, in the same fashion as the spoiler, with the split also satisfying the conditions of the join, Spoiler then picks either set 1 or set 2, and remove the other set of pebbles from both structures.

A *join game* on a pair of structures $(\mathcal{A}, \mathcal{B})$, is played as the regular 3 pebble game, except now the spoiler can use any number of $\bowtie$ moves, for $\bowtie$ in J. The winning conditions for both players are the same as in the 3-pebble game. For every recipe p of FO^3-join we also define the $L(p)$-join game. This contains all join games in which the sequence of moves performed by the spoiler are described by a path from the root of p to one of its leaves.

Let L be a class of FO^3-join formulae and A and B structures of vocabulary $\langle E, \sim \rangle$. We write $\mathcal{A} \preceq_L \mathcal{B}$ if $\mathcal{A} \models \varphi$ implies $\mathcal{B} \models \varphi$, for every sentence $\varphi \in L$.

Lemma 4. *The following are equivalent:*

- *The duplicator has a winning strategy on all $L(p)$ join games.*
- $\mathcal{A} \preceq_{L(p)} \mathcal{B}$

Before we prove this Lemma, we make the following crucial observation: If, in a join game a pebble has already been placed on element $a \in \mathcal{A}$, then the remainder of the game can be considered as a game with two pebbles on $(\mathcal{A}, a)$, until the first pebble is replaced somewhere else, or a join move are performed. We call these games *truncated.*

Proof of Lemma **??**. We prove the contrary: If there is a sentence φ of class $L(p)$ such that $\mathcal{A} \models \varphi$ but $\mathcal{B} \not\models \varphi$, then the spoiler has a winning strategy for the $L(p)$-join game.

We prove this by induction on the height of p.

The case when p is empty is trivial.

Assume that Lemma holds for all recipes of height k, and let p be a recipe of height $k+1$. Furthermore, assume that there is a sentence φ such that $\mathcal{A} \models \varphi$, but $\mathcal{B} \not\models \varphi$. We will construct a winning strategy for the spoiler. If φ is a boolean combinations of formulas, then the two structures are distinguished by at least one of them. We are thus left with the following cases:

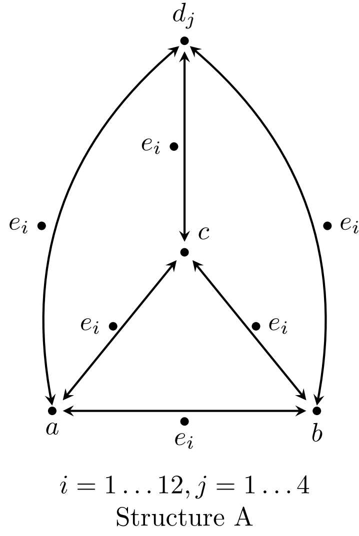
- φ is of form $\exists \psi(\bar{x})$, where $\bar{x}$ is a tuple of at most two variables, and ψ has depth at most $k-1$ and belongs to $L(q)$, where q is the subtree whose root is the single child of p. Then the spoiler can win as follows. In his first move he places one pebble in element a such that $(\mathcal{A}, a) \models \psi$. No matter in which element $b \in \mathcal{B}$ the duplicator places its pebble, we know that $(\mathcal{B}, b) \not\models \psi$, and thus the spoiler has a winning strategy for the remainder of the truncated game.
- φ is of form $\forall \psi(\bar{x})$, in which case the strategy is analogous to the previous one
- $\varphi(a,b,c)$ is of form $\varphi_1 \bowtie \varphi_2$, for some $\bowtie$ in J (note that a, b, c are interpreted as constants of A and B). Then p has two children p_1 and p_2, both of height $\leq k$, and $\varphi_1 \in L(p_1)$, $\varphi_2 \in L(p_2)$. Since $\mathcal{A} \models \varphi(a,b,c)$, yet $\mathcal{B} \not\models \varphi(a,b,c)$, spoiler can win by first placing pebbles on elements a, b, c, and splitting pebbles placing them into sets (a_1, b_1, c_1) and (a_2, b_2, c_2) of elements in A such that $(a_1, b_1, c_1) \bowtie (a_2, b_2, c_2) = (a, b, c)$. Given that $\mathcal{B} \not\models \varphi(a,b,c)$, then for every pair (d_1, e_1, f_1) and (d_2, e_2, f_2) of elements in B such that $(d_1, e_1, f_1) \bowtie (d_2, e_2, f_2) = (a, b, c)$, it must be the case that either $(\mathcal{B} \not\models \varphi_1(d_1, e_1, f_1)$ or $(\mathcal{B} \not\models \varphi(d_2, e_2, f_2)$. Depending on the move of the duplicator, spoiler chooses the set accordingly, and continues to win the truncated game on $(\mathcal{A}, a_i, b_i, c_i)$ and $(\mathcal{B}, d_i, e_i, f_i)$, for $i = 1$ or $i = 2$.

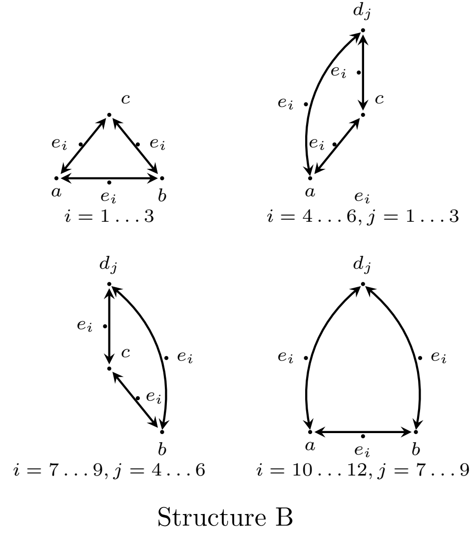
We now continue with the proof of the Theorem. Due to Lemma **??**, all that is left to do is to show structures A and B such that the duplicator can win any join game, and yet they are distinguished by an FO^4 formula.

The structures are as follows:

Consider objects a, b, c plus objects $d_1, \ldots, d_9$ and $e_1, \ldots, e_{12}$.

- Structure A contain edges $(a, e_i, b), (b, e_i, a), (a, e_i, c), (c, e_i, a), (b, e_i, c), (c, e_i, b)$, for each $1 \leq i \leq 12$, plus edges $(a, e_i, d_j), (d_j, e_i, a), (b, e_i, d_j), (d_j, e_i, b), (c, e_i, d_j), (d_j, e_i, c)$ for each $1 \leq i \leq 4$ and $1 \leq j \leq 12$.
- Structure B also has edges $(a, e_i, b), (b, e_i, a), (a, e_i, c), (c, e_i, a), (b, e_i, c), (c, e_i, b)$, for each $1 \leq i \leq 3$, plus edges $(a, e_i, b), (b, e_i, a), (b, e_i, d_j), (d_j, e_i, b)$ and $(a, e_i, d_j), (d_j, e_i, a)$ for each $1 \leq j \leq 3$ and for each $4 \leq i \leq 6$; $(a, e_i, c), (c, e_i, a), (d_j, e_i, c), (c, e_i, d_j)$ and $(a, e_i, d_j), (d_j, e_i, a)$ for each $4 \leq j \leq 6$ and for each $7 \leq i \leq 9$; and $(b, e_i, c), (c, e_i, b), (b, e_i, d_j), (d_j, e_i, b)$ plus $(c, e_i, d_j), (d_j, e_i, c)$ for each $7 \leq j \leq 9$ and for each $10 \leq i \leq 12$.





It is not difficult to see that the duplicator has a winning strategy for the standard 3-pebble games on this structure. If the three pebbles placed by the spoiler do not correspond with an edge of the structure, the the duplicator just mimics the same moves, the partial isomorphism trivially holds. If the third pebble correspond to some edge of form (u, e_i, v), for u and v in $\{a, b, c, d_1, \dots, d_9\}$ and $1 \leq i \leq 12$ in A that is not in B, assume the pebble was last placed in u (other two cases are symmetrical). Then the duplicator needs to find a permutation τ of the objects in A, such that $\tau(e_i) = e_i$, $\tau(v) = v$, $\tau(\mathcal{A})$ is isomorphic to $\mathcal{A}$ and the edge $(\tau(u), \tau(e_i), \tau(v))$ is in B, and place pebbles in $(\tau(u), \tau(e_i), \tau(v))$, so that the partial isomorphism still holds. For the remainder of the game, duplicator acts as if dealing with $\tau(\mathcal{A})$ instead of A.

Next, for the i, j, k-join move, assume that pebbles in structures $\mathcal{A}$ and $\mathcal{B}$ are in elements a_i, a_j, a_k and b_i, b_j, b_k, respectively. If spoiler divides first structure B duplicator just responds with the same edges in A. Now if spoiler divides structure A into pebbles (a_1, a_2, a_3) and $(a_{1'}, a_{2'}, a_{3'})$ satisfying the join condition, we have three cases:

- If none of (a_1, a_2, a_3) and $(a_{1'}, a_{2'}, a_{3'})$ are edges in A then duplicator mimics the pebble placement.
- If, say, only (a_1, a_2, a_3) is an edge in A, then the duplicator proceeds like in the above paragraph.
- Otherwise, if both (a_1, a_2, a_3) and $(a_{1'}, a_{2'}, a_{3'})$ are edges in A, duplicator needs to find a permutation τ of the objects in A such that $\tau(\mathcal{A})$ is isomorphic to A; $\tau(a_i) = a_i$, $\tau(a_j) = a_j$, and $\tau(a_k) = a_k$; and edges $(\tau(a_1), \tau(a_2), \tau(a_3))$ and $(\tau(a_4), \tau(a_5), \tau(a_6))$ belong to B, and respond with those pebbles. The partial isomorphisms trivially holds.

All that is left to show that this is a winning strategy for the duplicator is to show that there are always such permutations, no matter where are the pebbles placed. This can be easily shown with a lengthy and straightforward case by case analysis.

From Lemma **??** we obtain that A and B agree on all FO^3-join formulas. However, it is not difficult to see that they do not agree to the following FO^4 formula (which is only true in A):

$$\varphi(x, y, z) = \exists x \exists y \exists z \exists w \big(\psi(x, y, w) \wedge \psi(x, w, z) \wedge \psi(w, y, z) \wedge \psi(x, y, z) \wedge x \neq y \wedge x \neq z \wedge x \neq w \wedge y \neq z \wedge y \neq w \wedge z \neq w\big),$$

where

$$\psi(x, y, z) = \exists w \big(E(x, w, y) \wedge E(y, w, x) \wedge E(y, w, z) \wedge E(x, w, y) \wedge E(x, w, z) \wedge E(z, w, x) \wedge x \neq z \wedge x \neq y \wedge y \neq z\big).$$

This shows that FO^4 is not contained in TriAL.

□

Proof of Theorem 5

The containment of $\mathsf{TriAL}^=$ in FO^4 was shown in the proof of Proposition 4, and that $\mathsf{TriAL}^=$ contains FO^3 was already showed in the second part of the proof of Theorem 4, since the translation used there does not make use of inequalities in joins.

That the containments are strict follows from the proof of Theorem 4.

Proof of Theorem 6

Part 1

We begin by proving that TriAL^* is strictly contained in TrCl^6. To see that TriAL^* is contained in TrCl^6 we use induction on the structure of TriAL^* expressions. Note that all the cases, except for the Kleene closure of various joins we use, are precisely the same translation as in the proof of Theorem 4. What remains to prove is that expressions of the form

$$e' := (e \overset{i,j,k}{\underset{\theta,\eta}{\bowtie}})^*$$

can be translated into TrCl^6 expressions (the other join being completely symmetrical).

To see this, let $\psi_e(x, y, z)$ be a TrCl^6 formula equivalent to e. That is we have that $I_T \models \psi_e(a, b, c)$ if and only if $(a, b, c) \in R(T)$, for any triplestore T, with I_T the FO-structure representing T. We define the following formula $\psi_{e'}(x', y', z')$ in TrCl^6:

$$\psi_e(x', y', z') \vee \exists x, y, z \ \big(\psi_e(x, y, z) \wedge [\mathbf{trcl}_{x,y,z,x',y',z'} \varphi(x, y, z, x', y', z')](x, y, z, x', y', z')\big)$$

Where $\varphi(x, y, z, x', y', z')$ is a formula such that $\varphi(a, b, c, a', b', c')$ holds in I_T iff there exists a triple (a'', b'', c'') such that $\psi_e(a'', b'', c'')$ holds and the join of (a, b, c) and (a'', b'', c'') produces triple (a', b', c'). The definition of this formula in TrCl^6 is rather cumbersome, since it depends on the positions i, j, k of the join in question. We just give two examples, the rest are treated in the same way: For the expression $e' = (e \bowtie^{1,2,3'})^*$, we have that $\varphi(x, y, z, x', y', z')$ is $x = x' \wedge y = y' \wedge \exists x' \exists y' \big(\psi_e(x, y, z) \wedge \psi_e(x', y', z')\big)$. As another example, if $e' = (e \bowtie^{1',2',3'})^*$, then φ is just $\psi_e(x, y, z) \wedge \psi_e(x', y', z')$.

Next we prove that $\psi_{e'}$ is equivalent to expression e' over all triplestores. For one direction, let T be a triplestore database using a set O of objects, and assume that triple (a, b, c) belong to $e'(T)$. Then from the semantics of the recursive operator, there are sequences $t_1, \ldots, t_m$ of triples in O^3 and $p_1, \ldots, p_m$ of triples in $e(T)$ such that $t_1 \in e(T)$, and $t_{m+1} = t_m \overset{i,j,k}{\underset{\theta,\eta}{\bowtie}} p_m$. If $m = 1$ this follows from the first part of $\psi_{e'}$. If $m > 1$, notice that, by definition, $I_T \models \varphi(t_j, t_{j+1})$ for each $1 \leq j < m$. It follows that $I_T \models \psi_{e'}$. The other direction is analogous.

The fact that the containment is strict follows from **Part 3**.

Part 2

Next we prove that TrCl^3 is contained in TriAL^*. We do this by induction on TrCl^3 formulas. Note that all the cases, except for the case of transitive closure operator, are exactly the same as in the proof of Theorem 4. Next we show how to translate formulas of the form

$$\psi(x, y, z) := [\mathbf{trcl}_{x,y} \varphi(x, y, z)](u_1, u_2).$$

By the induction hypothesis there exists a TriAL^* expression R_φ such that for any triplestore T we have $I_T \models \varphi(a, b, c)$ iff $(a, b, c) \in R_\varphi(T)$.

Consider now the following expression R_ψ:

$$R := (R_\varphi \overset{1,2',3}{\underset{3=3' \wedge 2=1'}{\bowtie}})^*.$$

Observe now that a triple (a, b, c) will be contained in $R(T)$ iff there is a sequence of triples $(a, b_1, c), (b_1, b_2, c), (b_2, b_3, c), \ldots (b_k, b, c)$ with the property that they all belong to $R_\varphi(T)$. But this then means

that the pair (a,b) belongs to the transitive closure of the relation defined by $\varphi(x,y,c)$. That is we have that $(a,b,c) \in R(T)$ iff b is reachable from a using only edges defined by $\varphi(x,y,c)$.

We now proceed case by case, depending on the structure of terms u_1 and u_2. Since our terms are only variables we have a total of nine cases.

- If $u_1 = x$ and $u_2 = y$ we define $R_\psi := R$. It is straightforward to see that $(a,b,c) \in R_\psi(T)$ iff $I_T \models \psi(a,b,c)$.
- If $u_1 = y$ and $u_2 = x$ we define $R_\psi := R$.
- If $u_1 = x$ and $u_2 = z$ we define $R_\psi := \sigma_{2=3} R$.
- If $u_1 = z$ and $u_2 = x$ we define $R_\psi := \sigma_{1=3} R$.
- If $u_1 = x$ and $u_2 = x$ we define $R_\psi := \sigma_{1=2} R$.
- All of the other cases are symmetric.

This concludes the proof in the case when φ above uses x,y,z as variables. All of the other cases are similar, e.g. when we have the formula $[\mathbf{trcl}_{x,y}\varphi(x,y,x)](x,y)$ the expression $(\sigma_{1=3} R_\varphi \Join^{1,2',3}_{2=1'})^*$ in place of R will suffice (note that now we have only two free variables).

That the containment is strict follows from the comments at the beginning of the proof of Part 3 below.

Part 3

We begin by showing that TriAL^* is not contained in TrCl^4 or TrCl^5. In the proof of Theorem 4 we show that TriAL, and thus TriAL^* contain an expression e such that $e(T)$ is nonempty if and only if T has 6 different objects. The proof then follows by two classical results in finite model theory [26]: (1) e cannot be expressed by neither $\mathcal{L}^4_{\infty\omega}$ not $\mathcal{L}^5_{\infty\omega}$, the infinitary logic restricted to 4 and 5 variables, respectively, and (2) TrCl^k is contained in $\mathcal{L}^k_{\infty\omega}$

To see that TrCl^4 is not contained in TriAL (and thus that neither TrCl^5 not TrCl^6 are contained in TriAL), we define an analog of the logic FO^3-join used in the proof of Theorem 4. The logic FO$^3_\infty$-join extends FO^3-join with countably infinite disjunctions and conjunctions of formulas in FO^3-join (of course the restriction on the variables still holds). Formally, every FO^3-join formula is in FO$^3_\infty$-join, and if all φ_i are formulas in FO$^3_\infty$-join using the same set of at most 3 variables, for $i \in S$, where S is not necessarily finite, then $\bigwedge_{i\in S}\varphi_i$ and $\bigvee_{i\in S}\varphi_i$ are formulas in FO$^3_\infty$-join.

Notice that, by using these disjunctions, it is trivial to express the recursive star operator of TriAL^* with FO$^3_\infty$-join. Thus, if two structures $\mathcal{A}$ and $\mathcal{B}$ are indistinguishable by FO$^3_\infty$-join, then so are they by TriAL^*.

On the other hand, using the techniques in [26] it is not difficult to see that, if two structures $\mathcal{A}$ and $\mathcal{B}$ are indistinguishable by FO$^3_\infty$-join iff they are indistinguishable by FO^3-join (if the spoiler can win the join game on $\mathcal{A}$ and $\mathcal{B}$, then it can win the infinitary join game that characterizes FO$^3_\infty$-join).

It follows from the above observations, and the proof of Theorem 4, that TriAL^* cannot express the query

$$\varphi(x,y,z) = \exists x \exists y \exists z \exists w \big(\psi(x,y,w) \wedge \psi(x,w,z) \wedge \psi(w,y,z) \wedge \psi(x,y,z) \wedge x \neq y \wedge x \neq z \wedge x \neq w \wedge y \neq z \wedge y \neq w \wedge z \neq w\big),$$

where

$$\psi(x,y,z) = \exists w \big(E(x,w,y) \wedge E(y,w,x) \wedge E(y,w,z) \wedge E(x,w,y) \wedge E(x,w,z) \wedge E(z,w,x) \wedge x \neq z \wedge x \neq y \wedge y \neq z\big).$$

used in the proof of Theorem 4.

Lemma ??

The following lemma will be used several time in what follows.

Lemma 5. TriAL^* *is contained in the infinitary logic* $\mathcal{L}^6_{\infty,\omega}$.

What we mean by this is along the lines of the proof of Theorem 4, where we compare TriAL with first-order logic over the vocabulary $(E_1,\ldots,E_l,\sim)$.

That is to prove the lemma, we only have to show that the * operator can be simulated in this logic. To see this consider an arbitrary star-join of the form

$$R = (F \underset{\theta,\eta}{\overset{i',j',k'}{\Join}})^*.$$

Assume that we have an $\mathcal{L}^6_{\infty,\omega}$ formula $F(x_1, x_2, x_3)$ such that $T \models F(a, b, c)$ if and only if $(a, b, c) \in F(T)$. We first define the following formulas α, β. Consider the formula θ. We then let α be the conjunctions of formulas $x_i = x_j$, whenever $i = j$ is a conjunct in θ and $x_i \neq x_j$, whenever $i \neq j$ is a conjunct in θ. Similarly for $\rho(i) = \rho(j)$ in η we add $x_i \sim x_j$ as a conjunct in β and analogously for $\rho(i) \neq \rho(j)$.

We now define the following formulas:

- $R_1(x_1, x_2, x_3) := F(x_1, x_2, x_3)$.
- $R_{n+1}(x_1, x_2, x_3) := \exists x_4, x_5, x_6(R_n(x_1, x_2, x_3) \wedge \alpha \wedge \beta \wedge \exists x_1, x_2, x_3(x_4 = x_1 \wedge x_5 = x_2 \wedge x_6 = x_3 \wedge F(x_1, x_2, x_3)))$

Finally set $R(x_1, x_2, x_3) := \bigvee_{n\in\omega} R_n(x_1, x_2, x_3)$.

It is straightforward to check that this formula defines the desired relation over T. A similar formula can be defined for left-joins.

Note that we could have included constants to our comparisons with FO, but to keep the language one-sorted we omit them from our presentation. It is a straightforward exercise to check that all of the results would still hold true is they were allowed. For example constant comparisons of the form $2 = a$ would be handled by adding the clause $x_2 = a$ as a conjunct to the formula α above.

Proof of Theorem 7

Assume that GXPath uses a finite alphabet Σ of labels. We show that GXPath is contained in TriAL* by simultaneous induction on the structure of GXPath expressions. If we are dealing with a path expression α we will denote the TriAL* expression equivalent to α by E_α. Similarly when dealing with node expression φ, the corresponding TriAL* expression will be denoted E_φ. Note that for the node expression φ of GXPath we consider the TriAL* expression E_φ to be its equivalent if the answer set of φ is the same as the answer of $\pi_1(E_\varphi)$ over all graph databases and their triplestore representations, respectively.

Through the proof we will make use of the universal relation U containing all possible combinations of elements present in the model. We will also make use of the diagonal relation $D = U \bowtie^{1,1,1}_{1=1} U$ selecting all the triples (a, a, a) with $a \in V$.

Basis:

- $\alpha = a$ then $E_\alpha = E \bowtie^{1,2,3}_{2=a} E$
- $\alpha = a^-$ then $E_\alpha = E \bowtie^{3,2,1}_{2=a} E$
- $\alpha = \varepsilon$ then $E_\alpha = U \bowtie^{1,1,1}_{1=1} U$
- $\varphi = \top$ then $E_\varphi = U \bowtie^{1,1,1}_{1=1} U$

Inductive step:

- $\alpha' = [\varphi]$ then $E_{\alpha'} = E_\varphi \bowtie^{1,1,1}_{1=1} E_\varphi$
- $\alpha' = \alpha \cdot \beta$ then $E_{\alpha'} = E_\alpha \bowtie^{1,2,3'}_{3=1'} E_\beta$
- $\alpha' = \alpha \cup \beta$ then $E_{\alpha'}(x, y) = E_\alpha \cup E_\beta$
- $\alpha' = \alpha^*$ then $E_{\alpha'} = (E_\alpha \bowtie^{1,2,3'}_{3=1'})^*$
- $\alpha' = \overline{\alpha}$ then $E_{\alpha'} = E^c_\alpha$
- $\varphi' = \neg\varphi$ then $E_{\varphi'} = E^c_\varphi \cap D$
- $\varphi' = \varphi \wedge \varphi$ then $E_{\varphi'} = E_\varphi \cap E_\psi$
- $\varphi' = \langle\alpha\rangle$ then $E_{\varphi'} = E_\alpha \bowtie^{1,1,1}_{1=1} E_\alpha$.

It is straightforward to check that this translation works as intended. For illustration, consider the case when $\alpha' = \alpha \cdot \beta$. Our induction hypothesis is that we have two expressions, E_α and E_β such that (a, b) is in the answer to α on G iff $(a, c, b) \in E_\alpha(T_G)$, for some c and similarly for β. Assume now that (a, b) is in the answer to α' on G. Then there is c such that (a, c) is in the answer to α and (c, b) in the answer to β. But then $(a, c', c) \in E_\alpha(T_G)$ and $(c, b', b) \in E_\beta(T_G)$ for some c', b'. By the definition of join, we conclude that $(a, c', b) \in E_{\alpha'}(T_G)$. Note that all the implications above were in fact equivalences, so we get the opposite direction as well. All of the other cases follow similarly.

To prove that the containment is strict observe first that in [27], Theorem 4.3, GXPath was shown to be equivalent to $(FO^*)^3$, the three-variable fragment of FO with binary transitive closure [35]. Consider now the following TriAL

expression:

$$U \overset{1,2,3}{\underset{\varphi}{\bowtie}} U,$$

where $\varphi = (1 \neq 2) \wedge (1 \neq 3) \wedge (1 \neq 1') \wedge (2 \neq 3) \wedge (2 \neq 1') \wedge (3 \neq 1') \wedge \bigwedge_{a \in \Sigma, 1 \leq i \leq 3} i \neq a \wedge \bigwedge_{a \in \Sigma, 1' \leq i \leq 3'} i \neq a$ and U and U is the universal relation. It follows easily that this expression has an nonempty answer set if and only if the original graph database had at least four different nodes. It is well known that this query is not expressible in $\mathcal{L}^3_{\infty,\omega}$. Since $(FO^*)^3$ is contained in $\mathcal{L}^3_{\infty,\omega}$ we obtain the desired separation result.

Proof of Theorem 8

We begin by proving that full CNREs and TriAL^* are incomparable in terms of expressive power.

The existence of a CNRE query not expressible by TriAL^* simply follows from the fact that TriAL^* is contained in $\mathcal{L}^6_{\infty,\omega}$. The reason for this is that CNREs can ask for a 7-clique, a property not expressible in $\mathcal{L}^6_{\infty,\omega}$.

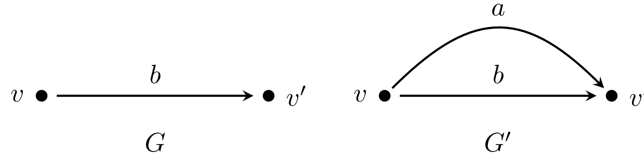
To see the reverse we will use a well know fact that CNREs are a monotonic class of queries. That is for any two graph databases G and G' such that $G \subseteq G'$ (that is G' contains all the nodes and edges of G) and any CNRE q we have that (u, v) is in the answer to q on G implies that (u, v) is in the answer to q on G' as well.

Next consider TriAL expression

$$e := (E \overset{1,2,3}{\underset{2=a}{\bowtie}} U)^c \overset{1,2,3}{\underset{\varphi}{\bowtie}} U,$$

with $\varphi = \bigwedge_{b \in \Sigma} 1 \neq b, 3 \neq b$. When interpreted over (a translation into a triplestore of) a graph database G, this expression returns all pairs of nodes that are *not* connected by an a-labeled edge. (Formally we will return all the triples u, v, w such that u and w are not connected by an a-labeled edge. The extra join just handles the specifics of our translation of a graph database into a triplestore). Suppose now that there is a CNRE q defining the aforementioned query.

Consider the following two graphs.



The nodes (v, v') will be in the answer to our query over the graph G. Using the monotonicity of CNREs and the fact that G is contained in G' we conclude that (v, v') is also in the answer to our query over G'. Note that this is a contradiction since we assumed that q extracts all pairs of nodes not connected by an a-labeled path.

This concludes the proof of part one of our Theorem.

Next we show that UCNREs using only three distinct variables are contained in TriAL^*. Observe first that for any NRE e there is a TriAL^* expression E_e equivalent to e over all data graphs (Corollary 2). We will now show that any CNRE that uses precisely three variables is definable using TriAL. To see this, consider the following example. Let Q be the following CNRE:

$$Q(x, y, z) := (x, e_1, y) \wedge (z, e_2, y) \wedge (y, e_3, y) \wedge (y, e_4, x).$$

It is easy to check that the following TriAL expression:

$$(((T_{e_1} \overset{1,2,3}{\underset{1=1}{\bowtie}} U) \overset{1,2,3}{\underset{2=2'}{\bowtie}} (T_{e_2} \overset{1,3,2}{\underset{1=1}{\bowtie}} U)) \overset{1,2,3}{\underset{2=2'}{\bowtie}} (T_{e_3} \overset{2,1,2}{\underset{1=3}{\bowtie}} U)) \overset{1,2,3}{\underset{2=2',1=1'}{\bowtie}} (T_{e_3} \overset{3,1,2}{\underset{1=1}{\bowtie}} U),$$

where T_{e_i} is the TriAL equivalent of e_i, is equivalent to Q over all graph databases.

Notice that here we have to output all the triples (x, y, z) satisfying the condition of our conjunctive query. For this we first join each T_{e_i} with the universal relation and arrange the nodes potentially appearing in the answer in the

right order. For example, when dealing with (x, e_1, y) we define $T_{e_1} \bowtie^{1,2,3}_{1=1} U$, where we put the nodes appearing in T_{e_1} in the correct order. At the end we simply join all the resulting relation in a way that preserves the designated objects. Here we have to take care that we force equality only on the objects used in the conjunctions involved up to now.

It is straightforward to extend this construction to the most general case of an arbitrary number of conjuncts with various arrangement of variables.

Finally, since TriAL expressions are closed under union we get that UCNREs with only three variables are contained in TriAL*. That the containment is proper it follows from the first part of the proof.

Proof of Corollary 4

The proof here follows the same lines as the one of Theorem 7. Because of this we only have to show how to define an equivalent TriAL* expression for any of the newly added data operators in GXPath($\sim$).

- For $\varphi = \langle \alpha = \beta \rangle$ we define $E_\varphi = E_\alpha \bowtie^{1,1,1}_{1=1',\rho(3)=\rho(3')} E_\beta$
- For $\varphi = \langle \alpha \neq \beta \rangle$ we define $E_\varphi = E_\alpha \bowtie^{1,1,1}_{1=1',\rho(3)\neq\rho(3')} E_\beta$
- For $\alpha' = \alpha_=$ we define $E_{\alpha'} = E_\alpha \bowtie^{1,2,3}_{\rho(1)=\rho(3)} E_\alpha$
- For $\alpha' = \alpha_{\neq}$ we define $E_{\alpha'} = E_\alpha \bowtie^{1,2,3}_{\rho(1)\neq\rho(3)} E_\alpha$

It is again straightforward to see that the described translations works as desired.

To show that the containment is strict we use a similar approach as when proving Theorem 7. We first notice that by combining the proofs of Theorems 4.3 and 6.2 from [27] one can show that GXPath($\sim$) is contained in $(FO^*)^3(\sim)$. Here by $(FO^*)^3(\sim)$ we denote the three variable fragment of FO^* enriched with the data value comparison operator $\sim$ that, when interpreted on data graphs, states if two nodes have the same data value. That is the formula $x \sim y$ will be true if and only if x and y have the same data value.

More formally, we will represent a data graph $G = (V, E, \rho)$ as a FO structure $G = (V, (E_a : a \in \Sigma), \sim)$ with $E_a = \{(v, v') : (v, a, v') \in E\}$. It is straightforward to see that with this interpretation we have GXPath($\sim$) $\subseteq (FO^*)^3(\sim)$.

It is also straightforward to see that the 3-pebble game [26] for $\mathcal{L}^3_{\infty,\omega}(\sim)$ follows the intended semantics when interpreted over data graphs. (Note that the game works over any class of structures, but over data graphs only relations are edge relations and the data value comparison.)

We can now play the 3-pebble game over the 3-clique graph and the 4-clique graph [26] where all data values are the same. The same winning strategy for the duplicator as in the game with no data values will still work, so we conclude that $\mathcal{L}^3_{\infty,\omega}(\sim)$ can not distinguish the two models.

Consider now the following TriAL expression:

$$U \overset{1,2,3}{\underset{\varphi}{\bowtie}} U,$$

where $\varphi = (1 \neq 2) \wedge (1 \neq 3) \wedge (1 \neq 1') \wedge (2 \neq 3) \wedge (2 \neq 1') \wedge (3 \neq 1') \wedge \bigwedge_{a\in\Sigma,1\leq i\leq 3} i \neq a \wedge \bigwedge_{a\in\Sigma,1'\leq i\leq 3'} i \neq a$ and U is the universal relation. It follows easily that this expression has different answer on the two models (since it asks for four different nodes in the original graph database). This finishes our proof.

Proof of Proposition 6

We begin by showing that register automata are not contained in TriAL*. To see this observe that it is straightforward to show that TriAL* is contained in the infinitary logic $\mathcal{L}^6_{\infty,\omega}$. This can be shown by a straightforward inductive translation.

Next we observe that for any n register automata can define a property not expressible in $\mathcal{L}^n_{\infty,\omega}$. For this consider the following regular expression with memory, shown in [28] to be equivalent to register automata:

$e_2 := \downarrow x_1 a[x_1^{\neq}] \downarrow x_2$
$e_{n+1} := e_n \cdot a[x_1^{\neq} \wedge x_2^{\neq} \wedge \cdots \wedge x_n^{\neq}] \downarrow x_{n+1}.$

Since no node can have more than one data value attached it follows that the answer to the query posted by the expression e_n is nonempty if and only if the graph database has at least n different elements.

It is well known [26] that $\mathcal{L}^n_{\infty,\omega}$ can not define a query stating that the model has at least $n+1$ element. Since TriAL^* is contained in $\mathcal{L}^6_{\infty,\omega}$ the desired result follows from the fact that e_7 is nonempty only on the graphs with at least 7 elements.

Next we prove that certain TriAL^* queries are not definable using register automata. To do so recall that register automata, when used as a graph query language, simply retrieve all nodes connected by a path whose label (i.e. the data word [28] it defines) belongs to the language of the automaton.

Consider now the query that returns all pairs of nodes such that there is no a-labeled edge between them. This query is easily seen not to be definable using register automata. Indeed, if it were, we could take any graph that has a nonempty answer set to this query and connect some two nodes in the answer set with an a-labeled edge. Now the same path that connected them in the original graph would work as a witness in the modified graph. This, however, is a contradiction, since the two nodes are not in the answer to the query on the new graph.

On the other hand, it is straightforward to define this query in TriAL using the expression $(\sigma_{2=a}E)^c$.