On the size of intermediate results in the federated processing of SPARQL BGPs

Jonas Halvorsen[a,b,*], Audun Stolpe[a]

[a]*Norwegian Defence Research Establishment (FFI), Postboks 25, 2027 Kjeller, Norway*
[b]*Department of Informatics, University of Oslo, Norway*

Abstract

This paper is a foundational study in the semantics of federated query answering of SPARQL BGPs. Its specific concern is to explore how the size of intermediate results can be reduced without, from a logical point of view, altering the content of the final answer. The intended application is to reduce communication costs and local memory consumption in querying dynamic network topologies and highly distributed, share-nothing or sharded architectures. We define row-reducing and column-reducing operations that, if a SPARQL resultset is viewed as a table, reduces the number of rows and columns respectively. These operations are deliberately designed so that they do not anticipate the unfolding of the evaluation process, which is to say that they do not presuppose knowledge about the structure or content of data sources, or equivalently, that they do not require data to be exchange in order to make intermediate results smaller. In other words, the operations that are studied are based solely on the shape of evaluations trees and the distribution of variables within them. The paper culminates with a study of different compositions of the aforementioned reduction operators. We establish mathematically that our row- and column operators can be combined to form a single reduction operator that can be applied repeatedly without altering the semantics of the final result of the query answering process.

Keywords: Federated query processing, intermediate results, minimization, blank nodes, sparql

1. Introduction

Federated SPARQL processing concerns the task of answering a a global query using the combined information from distinct sources. It involves breaking up a global query into a set of jointly exhaustive subqueries each of which is directed to a particular SPARQL endpoint before the results are returned to the federated query processor and combined into a correct answer to the initial global query, if one is to be had. That the exploitation and dissemination of Semantic Web data requires powerful federation engines is something of a truism, given the Web wide scope of names in RDF and the whole Linked Data philosophy.

This paper formalizes and investigates various optimizations that can be used to lighten the overall dataflow that this process consumes. More specifically, it is concerned with the question of how to reduce the size of intermediate result without compromising the semantics of the final answer of said global query.

Although the problem of keeping intermediate results small is of interest to both local and federated query execution, it is particularly pressing in the distributed case where the triples participating in a join may be stored on different servers. Such *cross-site* joins require network communication during join evaluation, that is, data has to be exchanged between servers in order to evaluate the join in question. Needless to say, this will claim bandwidth and CPU time proportionate to the amount of data that is exchanged, and as pointed out in [1] may easily grow with the overall data size to exceed the capacity of individual servers. Hence if the size of intermediate results is allowed to grow unconstrained, then in addition to any capacity issues with bandwidth and/or remote servers, it is likely that memory overflow problems will propagate back to the local thread of execution. Therefore, how *little* data one can send and keep in memory without sacrificing the precision and completeness of the final query answer should be a worthwhile question to address.

We approach this question by studying combinations of reduction operators, as we shall call them, in different order. Some of these operators are best regarded as part of the folklore, although we believe we offer at least one new one as well. However, the main contribution of the present paper is an integrated formal account of these operators that allows them to be studied in combination in a mathematically principled manner.

There are two kinds of reduction operators: operators that remove redundant rows and operators that remove redundant columns. We are after the conservative cores, so to speak, of intermediate results, by which we mean the smallest amount of data that needs to be retained to prevent information loss in the final query answer.

The first hurdle here is to clarify what it means for a federated SPARQL processor to lose information. Our take on this is to say that a federated SPARQL processor should return the same answer *set* (to make life a bit easier, we adopt the set semantics rather than the multitset semantics for SPARQL) as

*Corresponding author
Email addresses: jonas.halvorsen@ffi.no (Jonas Halvorsen), audun.stolpe@ffi.no (Audun Stolpe)

the one that would be returned were the query to be executed against the *merge*, in the technical sense of [2], of the contributing RDF graphs. Of course, this passes the buck to the concept of sameness, and since equality is in some cases too much to ask, we shall have to be explicit about the conditions under which we regard two answer sets as equivalent.

The concept of equivalence should be robust enough to allow a reduction operator to be applied in a second pass without changing the semantics of the final answer, for as it turns out, some reduction operators give rise to new redundancies that show up after the reducts are combined. Since we want intermediate results to be as small as possible, we should be allowed to make a second pass to remove the surplus.

At this point, the conceptual situation is already quite complex. To bring some order to the investigation, therefore, we start the formal development by summarizing our requirements in an abstract characterization of the concept of a reduction operation as we construe it. This characterization serves a dual purpose: first it makes it perfectly clear how the concepts of losslessness (aka. answer set completeness), answer set equivalence, and reduction of intermediate results are connected. Secondly, and more interestingly, the abstract characterization constitutes a stratum that allows the interaction between reduction operators to be studied in a principled manner: complex reduction operators can be formed by composing elementary ones and checking that the result satisfies the abstract definition. One of the novel things to come out of this, we believe, is the finding that not all compositions are equally effective, some will produce smaller intermediate results than others.

Three elementary reduction operators are selected for closer study. Some of the underlying ideas will have a familiar ring to them as they sometimes trade on themes that recur in database theory in one form or another. We shall try to indicate the connections as we go. The operators in question can be described informally as follows: the first is a projection pushing operator that removes columns when they no longer contribute, typically by providing join arguments, to the evaluation process. The second is the operation that removes rows if they contain blank nodes in join position. This heuristic is based on a result from [3] which entails that *any* federated evaluation tree that produces a correct answer can be assumed to be of a form that makes intermediate results disjoint wrt. blank nodes unless they lie on the same branch. To the best of our knowledge, this simple but generally applicable reduction operator is a novel contribution of the present paper. The third and final operation is based on the relation of *informativeness* between answer sets. It is an adaptation of a concept from database theory explored by Libkin in [4, 5], though we modify it and put it to a quite specific use. Briefly put, we use the informativeness relation to select cardinality-minimal but equivalent subsets of an answer set. We think of the former as compressions or *kernels* of the input set, and show that the operation taking an answer set to its kernel is a reduction operation.

An outline of the paper, and a summary of its contributions goes as follows: we give an abstract characterization of reduction operations in general in Section 6. In Section 7 we define a projection pushing operation, or *truncation* as we prefer to call it, that is not in itself new. What *is* new, besides the operator formulation of it, is this: first we demonstrate that the truncation operator adheres to the abstract pattern of a reduction operator. Secondly, we prove that truncation is optimal in the sense that, given certain natural provisos, no column-reduction operation yields smaller intermediate results. Next, in Section 8 we study the operation of removing rows with blank nodes in join positions from intermediate results and show that that too constitutes a reduction operation in our sense. Although it is a rather obvious operation, we believe that the general applicability of it, that is, that fact that it is a reduction operation that may be applied across the board to all federated SPARQL evaluation trees without loss of results, is established for the first time in the present paper. In Section 9 we formulate the concept of informativeness and apply it to identify cardinality minimal equivalent subsets of an answer set. As mentioned already, whilst the relation of informativeness is not new, we believe this particular application of it is. As the reader should by now have come to expect, we also prove that it is a reduction operation in the abstract sense. Section 10, the final substantial section of this paper, is largely example driven, although it opens with a couple of corollaries that state that compositions of the elementary reduction operations yield reduction operations. The examples are designed to show that the situation wrt. combining reduction operations is rather subtle and multi-faceted. Two general lessons can be learned: first, ordering matters; some compositions produce smaller answer sets than others. Just how small is a question we leave for future research. That is we do not offer a minimality result for any of the combined operations similar to that for the truncation operator, although we can say certain things about which combinations are *not* minimal. The second lesson to be learned is that when row-reduction and column-reduction operations are made to act in consort, they reduce intermediate results beyond the threshold of the row operation acting alone. That is, the reduct of an answer set under the complex operation will contain fewer rows than the reduct of the same set under only the row operation.

2. Related Work

A substantial amount of research has been produced that is relevant for reducing intermediate results in federated SPARQL processing.

One line of research is that of algebraic optimization of SPARQL queries. Here, algebraic equivalences are used to rewrite queries into ones that can be computed more efficiently. In [6], algebraic laws for projection pushing and filter manipulation for SPARQL are given. The relation to the present work is first and foremost when it comes to the matter of projection pushing, which corresponds to the operation we call *truncation*.

Optimization rules are also studied in [7], in the context of the SERVICE operator of the SPARQL 1.1 federation extension and the interplay with OPTIONAL patterns. This is not directly comparable to the present work, since only basic graph patterns are allowed hence neither OPTIONAL or SERVICE operators, or any similar operators are catered for. However, that work is generalized to SPARQL queries without the use of the SERVICE operator in [8], presenting algebraic equivalences for SPARQL federated queries that utilize shipping of intermediate results through the use of either the SPARQL VALUES or FILTER operators in order to reduce intermediate results. The same paper further presents a rewriting algorithm that performs filtering of blank nodes in the shipped values, based on overlapping variables. However, this process is applied stepwise from one node to the next in the evaluation tree rather than holistically for the whole evaluation tree as such. Hence solutions with blank nodes in join positions might linger in intermediate results until the evaluation process reaches the point in the evaluation tree where the relevant join occurs. Thus, intermediate results are not as small as they can be, and not as small as those produced by the operators described in this paper.

Taking stock, the main difference between the abovementioned line of work, is that in this paper, the focus is on studying different combinations of generic reduction operators, and how they affect results rather than the study of algebraic rewriting rules for the query language. For instance, the truncation operation that is studied in Section 8 of the present paper yields the same results as the SPARQL-algebraic projection pushing technique outlined in [6], but within a logical framework for studying answer preservation in a federated setting.

Another line of relevant research is that of query plan optimization. A paradigmatic case is that of finding an optimal join ordering based on selectivity estimates for the leaves [9]. These approaches typically focus on coining cost functions for determining selectivity of triple patterns, and are based either on general heuristics regarding the structure of triple patterns ([10]) or cost functions generated from concrete datasource statistics ([11] and [12]). Furthermore, there is a substantial amount of work on rewriting the query plan based on grouping triple patterns together by variable-counting and aggregated sums based on previously mentioned cost functions ([13] and [12]). The present work is best seen as being orthogonal to both join-order optimization and grouping of triples. More concretely, the approach outlined in this paper is agnostic wrt. the form of a particular evaluation tree, that is wrt. its structure understood as the selectivity of patterns it contains and its join order.

Yet another branch of optimizations of relevance is based on restrictive source selection. Here, the idea is to avoid overestimating the number of sources that need to be included in the evaluation, as this incurs more network traffic than necessary. Some approaches, such as [14], assume that the data is cleanly partitioned into sources, assuming that sources do not share vertices. Other approaches rely on knowledge regarding triple duplication ([15] and [16]) or join-awareness ([17] and [18]) through the use of indexes or other coordination mechanisms, in order to reduce intermediate results being transferred. Either way, these approaches presupposes knowledge as to where the concrete data occurs, something that our approach is assumed not to have access to.

With respect to both source selection and query plan optimization, the work in this paper introduces operations that given *any* query plan can be applied to the nodes in that tree in order to produce their conservative cores. It is therefore fully compatible with approaches that seek to e.g. limit network communication by selecting sources wisely or reduce payloads by leveraging semi-joins.

There is also a general difference of nature between this paper and all the mentioned related work. That is, we study reduction operators adhering to a certain mathematical framework for answer preservation that ensures that the final answer to the global query is to stay the same. The concept of sameness that is appealed to here will be defined in due course, and is to the best of the authors' knowledge a novel contribution. Secondly, in order to identify what may be thought of as conservative cores of intermediate results, the present paper pays particular attention to and leverages the semantics of blank nodes in SPARQL semantics. The topic of blank nodes in SPARQL federation has to a large extent been neglected—with blank nodes usually being ruled out by assumption—but as shown [3], it is a significant one.

3. Preliminaries

3.1. Nomenclature

Conventions. For notational economy curly braces will be omitted from singletons in set-theoretic expressions as well as from arguments of functions if no confusion is likely to ensue, e.g. $P \cup t$ instead of $P \cup \{t\}$ and $f(t)$ instead of $f(\{t\})$. Also, when f is a function and A a subset of f's domain, then $f(A)$ is shorthand for the set of elements b such that $b = f(a)$ for some $a \in A$. If f is a function, $dom(f)$ and $ran(f)$ are its domain and range respectively.

RDF graphs. Let I, B and L denote pairwise disjoint infinite sets of IRIs, blank nodes, and literals respectively. In conformity with the nomenclature of [19], IL abbreviates $I \cup L$ and T abbreviates $I \cup B \cup L$. T is the set of *RDF terms*. IRIs will be denoted by lower case letters prepended by colons, e.g. $:s$ or $:d$, whereas a blank node will have an additional underline in front of it, e.g. $\_:b$. An RDF *triple* (or just 'triple') is an element $t \in IB \times I \times IBL$. An RDF *graph* is a finite set of RDF triples. RDF graphs are denoted by possibly subscripted Gs, and sets of RDF graphs by $\mathscr{G}$s.

SPARQL queries. Turning now to SPARQL queries, $\mathcal{V}$ symbolizes an infinite set of variables disjoint from IBL. Individual SPARQL variables will be denoted by lower case letters from the end of the alphabet prepended by question marks, e.g. ?x and ?z. A SPARQL *triple pattern* (or just 'triple pattern') is an element $\in IL\mathcal{V} \times I\mathcal{V} \times IL\mathcal{V}$. We shall let the notation t do dual service and denote both SPARQL triple patterns and RDF triples. We are thus deliberately blurring the distinction between them. This is mathematically convenient, for reasons that will become clear as we go, and we shall usually rely on context to disambiguate. A *conjunctive SPARQL pattern*, symbolized by a possibly subscripted P, is a set of triple patterns. A conjunctive SPARQL query (or just'conjunctive query') is a pair (W, P) where $W \subseteq \mathcal{V}$ and P is a conjunctive SPARQL pattern.

Set semantics of conjuctive queries. The set semantics of conjunctive queries, is defined by a function that interprets a conjunctive query in an algebra of all sets of solutions. A *solution*, in turn, is a partial function $\mu : \mathcal{V} \to IBL$. Two solutions μ_i and μ_j are *compatible*, written $\mu_i \rightleftharpoons \mu_j$, if their union is a partial function. The set of all solutions is denoted Σ. A subset $\mathscr{A} \subseteq \Sigma$ is an *answer set* if all solutions have the same domain. We let $\mu(t)$ stand for the result of uniformly substituting RDF terms for variables in t according to μ. Note that $\mu(t)$ is well-defined irrespective of whether the domain of μ contains all variables in t or not: if it does then $\mu(t)$ is a triple, if not then $\mu(t)$ is a triple *pattern*.

A conjunctive SPARQL algebra is a structure $\langle 2^{\Sigma}, \bowtie, \pi \rangle$, where $\bowtie$ is a binary operator defined as

$$\Omega^i \bowtie \Omega^j \quad := \{\mu_i \cup \mu_j \mid \mu_i \in \Omega_i, \mu_j \in \Omega^j, \mu_i \rightleftharpoons \mu_j\}$$

for $\Omega^i, \Omega^j \in 2^{\Sigma}$, and. The operation π is a function of type $2^{\mathcal{V}} \times 2^{\Sigma} \mapsto 2^{\Sigma}$ defined as:

$$\pi_W(\Omega) \quad : = \{\mu_{|(W \cap dom(\mu))} \mid \mu \in \Omega\}$$

where $\mu_{|X}$ denotes the restriction of μ to X.

A conjunctive query $Q := (W, P)$ is evaluated over a graph G by an interpretation function $[\![]\!]_G$ that maps queries into the SPARQL algebra in the following manner:

$$\begin{array}{ll} [\![t]\!]_G & := \{\mu \in \Sigma \mid dom(\mu) = vars(t), \mu(t) \in G\} \\ [\![P_i . P_j]\!]_G & := [\![P_i]\!]_G \bowtie [\![P_j]\!]_G \\ [\![(W,P)]\!]_G & := \pi_W([\![P]\!]_G) \end{array}$$

If Q is a conjunctive query then $[\![Q]\!]_G$ will be called an *answer to Q over G*. The analysis in the present paper will be restricted to this fragment of the SPARQL language.

For a *set* of graphs $\mathscr{G}$, the evaluation $[\![P]\!]_{\mathscr{G}}$ of P is understood to be the *federated evaluation* of P over $\mathscr{G}$, which is defined to be the union of the results of evaluating P over each $G \in \mathscr{G}$ (cf. [3]). In contrast, let $m(\mathscr{G})$ denote the *merge* of the graphs in $\mathscr{G}$, that is, $m(\mathscr{G})$ is the single graph that results from taking the union of all elements of $\mathscr{G}$ after standardizing apart blank nodes from different graphs. Then $[\![P]\!]_{m(\mathscr{G})}$ is just the evaluation of P over the single source $m(\mathscr{G})$. The two are emphatically not the same. Indeed, the semantics of federated zero-knowledge query processing is precisely the conditions under which $[\![P]\!]_{\mathscr{G}} = [\![P]\!]_{m(\mathscr{G})}$. The left-to-right inclusion says that the federation process is sound, and the converse inclusion says that it is complete.

3.2. Trees

Let A^* denote the set of strings over the alphabet $A := \{1, 2\}$ A *tree domain* is a subset D of A^* satisfying the conditions

1. For each $k \in D$, every prefix of k is also in D.

2. For each $k \in D$, $k2 \in D$ iff $k1 \in D$.

Every tree domain can be ordered by the prefix ordering $\leq$ on binary strings. Supervenient on this ordering we define an *evaluation tree* as a total function Ψ from D to $\mathscr{A}$ satisfying the condition

$$\Psi(n1) \bowtie \Psi(n2) = \Psi(n) \tag{1}$$

The set of evaluation trees will be symbolized by $\mathbb{T}$.

The function Ψ can be viewed as an indexing function on elements of $\mathscr{A}$, whence pairs $(i, \Omega) \in \Psi$ can be interpreted as indexed sets Ω^i. We shall usually prefer the latter to the former notation. The *root* of a binary operator tree Ψ is the answer set Ω^{ϵ}. Given a tree Ψ and an index k in $dom(\Psi)$, the *subtree rooted at i*, written Ψ/i, is the tree whose domain is the set $\{m \mid im \in dom(\Psi)\}$ and such that $(\Psi/i)(n) = \Psi(in)$ for all $n \in dom(\Psi/i)$. The set of *leaves* in Ψ is denoted $l(\Psi)$. The *depth* of a tree Ψ is the longest path from the root to a leaf, equivalently it is the length of the longest index in $dom(\Psi)$. These definitions ultimately go back to [20]. We extend the notation *var* to trees and write $var(\Psi)$ for the set of SPARQL variables that occur in the domain of some μ in some Ω in $l(\Psi)$.

4. The problem

The general theory of the federated evaluation of conjunctive SPARQL queries was developed in [3] and [21]. The theory being general means that the it allows arbitrary occurrences of blank nodes in the data without compromising the soundness or completeness of query answers. The interested reader should consult [3] and [21] for the details. In this section we shall be content to show by example what special problems federation across blank nodes presents, and to sketch what requirements this imposes on a sound and complete query processor.

For the purposes of the present paper, the point of this is to highlight a couple of key properties that we are allowed to assume for *all* evaluation trees combining results from multiple execution contexts. Due to the semantics of blank nodes, these assumptions are valid for federated SPARQL processing in general. These properties present certain heuristic opportunities that form the basis and governs the interaction between the reduction operations that are studied in the remainder of the paper.

4.1. A motivating example

A natural default requirement for a federated SPARQL processor is that it should return all the solutions to a query that is warranted by the union of the RDF graphs that that query is federated over. For illustration, consider the two RDF graphs in Figs. 1 and 2 respectively. These graphs encode information regarding members of the European Parliament (MEP), as found in the LinkedEP dataset produced by the Talk of Europe project [22], a dataset covering plenary debates held as well as biographical information regarding members of parliament. More specifically, source A encodes information regarding the MEP Eva Joly and her political functions, while source B encodes information regarding MEP Carl Schlyter. From the data, we see that they represent different national parties but belong to the same EU political party (Europarty). However, the information in source A alone is *not* enough to conclude that Eva Joly is associated with a Europarty, as EFA is not typed as such. This missing piece of information is, however, present in source B. Thus, when the sources are merged, as shown in Fig. 3, the political institutions are all appropriately typed. Hence, posing the query in Lst. 1, asking for the name of the MEPs in the EU parliament that are politically affiliated with a Europarty (not all MEPs are), as well as the party name, produces the answers in Fig. 5.

```
SELECT ?person ?party  WHERE {
  ?person a lpv:MEP.
  ?person lpv:politicalFn ?x.
  ?x lpv:institution ?party.
  ?party rdf:type lpv:EUParty.}
```

Listing 1: Get MEP and EU party

Now, if we only evaluate the query in Lst. 1 against each source separately, for so to take the union of the results, we get an incomplete set of answers as shown in Fig. 4. In other words, it is clear that the sum of the whole is more than the sum of its separate parts. That is, the total amount of information contained by the two sources combined, resides not only in what each of them can contribute separately, but in also in the combination or join of elements across sources. In other words, the query cannot simply be executed as a whole against each source—that is too coarse. It must rather be split up into parts tailored to capture the cross-site joins.

Unfortunately, there is a complicating factor that blocks any straightforward realization of this idea, namely the presence of blank nodes in join positions. More specifically, sources A and B utilize blank nodes to represent complex attributes in the form of statements about statements, as recommended by the Semantic Web Best Practices and Deployment Working Group. In this case, that "X had a political affiliation to institution Y between dates A and B" is codified using blank nodes. In the distributed case, such a join, if it is not handled with special care, will quickly become a drain through which information will leak. As described in detail in [3], this is due to the fact that anaphoric reference is lost whenever the same blank node is processed in two separate execution contexts. According to the SPARQL 1.1 specification, every distinct query constitutes a distinct and sealed scope for blank node identifiers, which means that a blank node from one execution context cannot be referenced in another. Blank nodes are similar to existential variables in the sense that they are anaphors within the same quantificational context only. Now, a blank node that receives different names in different query execution contexts obviously cannot be used for cross-site joins, so there it is.

It is worth emphasizing that none of the more straightforward and better known query-decomposition strategies from the literature, such as the *even decomposition*, so called in [21] as implemented in DARQ [23], and the *standard decomposition* as implemented in FedX [13] solve this problem.

Exemplifying, the *even decomposition* will evaluate each triple pattern (from the *global* query, let's call it) against every source that *may* contain an answer for it (meaning that the RDF property from the triple pattern in question occurs in that source). For instance, the even decomposition will evaluate both of the triple patterns `?person lpv:politicalFn ?x` and `?x lpv:institution ?party` from the query in Lst. 1 separately against each of A and B. Collecting the solutions in separate tables, we have the answer sets in Figs. 6 and 7, where the identifiers for blank nodes have been given distinct subscripts c and d to signify that they are not to be treated as the same names. Now, as these tables do not join, the even distribution produces no answer to the example query, not even the ones that derive from the same source. This time it comes down to the fact that query is split too finely.

Taking stock, these examples can be taken to show the following: If answering a query involves joins on blank nodes, then the granularity of the decomposition of that query matters a great deal. If the query is split too finely, then answers from a single source may be lost due to the loss of join information linking the partial answers. If on the other hand the query is split too coarsely, then cross-site joins may be lost.

5. Properties of correct decompositions

Reasoning formally about federated evaluation of conjunctive SPARQL queries requires a minor amendment to the semantical apparatus introduced so far: when different portions of a query are directed to different SPARQL endpoints they are also evaluated in different execution contexts. According to the specification they should therefore not share blank nodes between them. We need to make sure that this disjointness condition is properly maintained, which is why we introduce a parameter c in $[\![P]\!]^c_G$ as an explicit representation of a particular execution context. Mathematically, it is a relabeling function that ensures that for each execution context blank nodes are given identifiers that belong uniquely to that context. Solutions in $[\![P]\!]^c_G$ will accordingly be denoted μ^c, though the index c may be omitted when it is clear from context. See [3] for mathematical details.

Going back to the example from the preceding section, the partition immediately below gives a decomposition of the query in Lst. 1 that produces a correct and complete answer.

P_1 := {`?person lpv:politicalFunction ?x.,`
`?x lpv:institution ?party.`}
P_2 := {`?person a lpv:MEP.`}
P_3 := {`?party a lpv:EUParty.`}

The reason that this decomposition succeeds where the standard and even decompositions fail is first, that it groups together those triple patterns that match a join on a blank node thus ensuring that joins on blank nodes are evaluated in a single execution context (P_1). Secondly, all other triple patterns are shipped as singletons which prevents *cross-site* joins *not* involving blank nodes from being lost (P_2 and P_3).

As it happens, there is only one solution μ to the query in Lst. 1 over the sources in Figs. 1 and 2. The decomposition P_1-P_3 corresponds to this solution, meaning that the join of the respective unions of evaluating each subquery over the source it is assigned to yields the only correct answer.

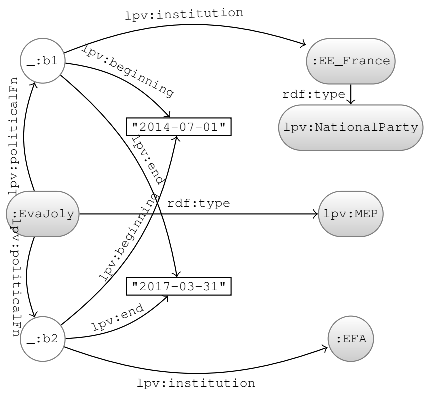


Figure 1: RDF source A

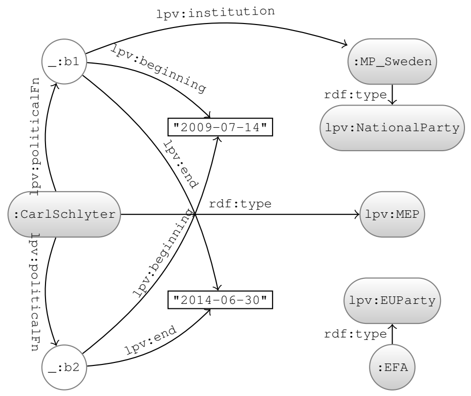


Figure 2: RDF source B

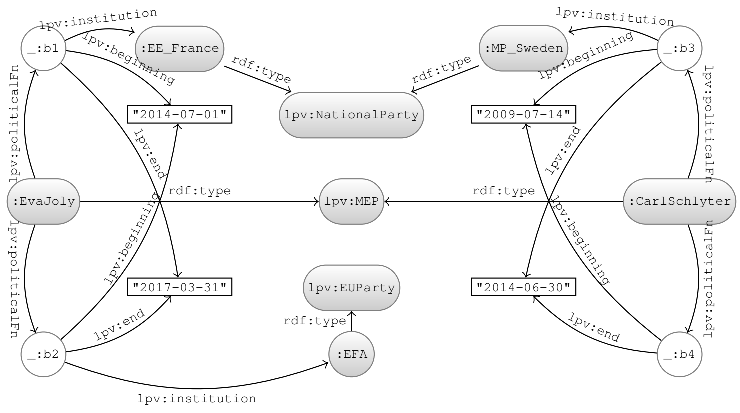


Figure 3: The union of sources A and B modulo renaming of blank nodes.

?person	?party
:CarlSchlyter	:EFA

Figure 4: Union of answers over A and B

?person	?party
:EvaJoly	:EFA
:CarlSchlyter	:EFA

Figure 5: Answer over the merge of A and B

?person	?x
`:EvaJoly`	$\_:b1_c$
`:EvaJoly`	$\_:b2_c$

Figure 6: `?person lpv:politicalFn ?x` over A

?x	?party
$\_:b1_d$	`:EE_France`
$\_:b2_d$	`:EFA`

Figure 7: `?x lpv:institution ?party` over A.

In the general case that a query has more than one solution ('having a solution' should here be understood as having an answer in the *merge* of the contributing sources) different decomposition may be required. Indeed, it is not entirely obvious that there is a decomposition for every solution. The demonstration that there is, relies on the concepts of a *b-component* and a *b-connected* set:

Definition 5.1 (*b*-connectedness)**.** Let G, $\{a\}$ be RDF graphs, then

1. $\{a\}$ is b-connected
2. $G \cup \{a\}$ is *b*-connected if G is *b*-connected and G and a share a blank node.

∘

A *b-component* is a subquery that matches a maximally *b*-connected subgraph modulo some solution μ:

Definition 5.2 (*b*-component)**.** Let $\mu_c \in [\![P]\!]^c_{m(\mathscr{G})}$ and suppose $P_i \subseteq P$. Then P_i is a *b*-component of P relative to μ^c iff $\mu^c(P_i)$ is a maximal *b*-connected subset of $\mu^c(P)$.

∘

Note that *b*-connected sets are RDF graphs, whereas *b*-components are SPARQL query patterns. Note also that subquery P_i is a *b*-component *relative to* a particular solution μ. We shall say that μ *induces* the *b*-component P_i.

Now, let μ^c be a solution to P in a graph G and let $f(\mu^c, P)$ denote the set of *b*-components of P modulo μ^c. Then f is a function and $f(\mu^c, P)$ partitions P. Indeed $f(\mu^c, P)$ selects the partition that corresponds to μ^c.

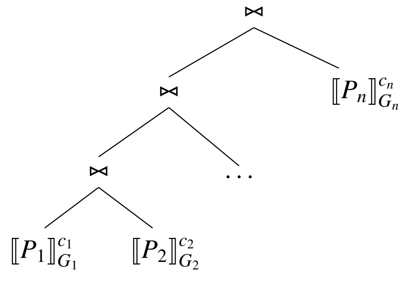
Theorem 5.1. *Let $\mathscr{G} := \{G_i\}_{i \in I}$ be a set of sources of RDF graphs and let $\mu^c \in [\![P]\!]^c_{m(\mathscr{G})}$. Put $f(\mu^c, P) := P_1, \ldots, P_k$. Then there is a set $\{m, \ldots, n\} \subseteq I$ such that there is a $\mu' \in [\![P_1]\!]^{c_m}_{G_m} \bowtie \ldots \bowtie [\![P_k]\!]^{c_n}_{G_n}$, for any distinct set of execution contexts $c_m, \ldots, c_n$, and $\mu^c(P) \dashv\vDash \mu'(P)$.*

Proof. This is Corollary 5.5 in [3]. □

Theorem 5.1 shows that if there is a solution to P in the merge of the RDF graphs $\mathscr{G}$, then there is an equivalent solution of form $f(\mu^c, P)$ that can be assembled by *federating* P over $\mathscr{G}$. All these solutions are alphabetic variants of each other, obtained by substituting names of blank nodes for names of blank nodes. Moreover, the completeness theorem of [3] (Theorem 7.1) shows that *every* solution obtained by federation has this form, i.e. is $f(\mu^c, P)$ for some μ^c.

The importance of this is that it allows us to assume that if a decomposition of a query, and thus by extension an evaluation tree, produces a solution at all, then it will have the property that exactly one subquery is evaluated over each RDF graph. Contrapositively, if it is not of that form, and the blank nodes are properly distinguished by execution contexts, then it returns the empty solution. More specifically, the subqueries in $f(\mu^c, P)$ will be such that all joins on blank nodes are *contained* within them, meaning that blank nodes are never split between execution contexts. One way of looking at this is to view blank nodes as indexes into sources: whenever two blank nodes can be identified they must stem from the same source. Hence, if there is a legitimate join on a blank node then both arguments to that join can safely be assumed to be collocated.

For the purposes of formulating information preserving reduction operators there is no reason to worry about decompositions that return empty sets, and so if the conjunctive SPARQL query being evaluated is (W, P) then any distribution of it will be assumed to have the following form. See Fig. 8.



$\bowtie$
$\bowtie$
$[\![P_n]\!]^{c_n}_{G_n}$
$\bowtie$
...
$[\![P_1]\!]^{c_1}_{G_1}$
$[\![P_2]\!]^{c_2}_{G_2}$

Figure 8: The form of distributed evaluation trees.

For easy reference we extrapolate and name a pair of consequences of this demarcation of the set of eligible evaluation trees:

The separation assumption: None of the leaves in a distributed evaluation tree share blank nodes, and hence

The idle join assumption: A blank node in join position can not be, and never needs to be, combined with any solution from any other partial answer unless they are on the same path.

6. An abstract characterization of reduction operators

Intuitively a reduction operation, in the sense intended in the present paper, is an operation that can be applied to intermediate results to make them smaller, but without interfering with the semantics of the final result of the query. Stated in terms of evaluation trees, a reduction operator should be applicable to all the nodes of an evaluation tree, it should produce smaller nodes and it should preserve the semantics of the final answer up to some plausible notion of equivalence.

Moreover, the outputs of the reduction operator—we shall call them reduced sets (or, nodes, depending on context)—must be related in the correct manner. Specifically, root of the reduced tree must be generated from the reduction of the leaves in a process that interleaves the join operation with the reduction operation in the right manner. Definition 6.1 furnishes a reduced evaluation tree with the requisite recursive structure.

Definition 6.1 (Reduced evaluation tree)**.** Let o be an operation of type $o : \mathbb{T} \times A^* \times \mathscr{A} \longrightarrow \mathscr{A}$. In practice, the first two arguments will always be fixed and o will consequently be treated as a one-place operation o^Ψ_i. The *reduct* Ψ_o of an evaluation tree Ψ, is a tree (not necessarily an evaluation tree) derived from Ψ in the following manner:

$$\Omega^i_o =_{df} \begin{cases} o^\Psi_i(\Omega^i) & \text{if } \Omega^i \text{ is a leaf} \\ \\ o^\Psi_i(\Omega^j_o \bowtie \Omega^k_o) & \text{if } \Omega^i = \Omega^j \bowtie \Omega^k \end{cases}$$

◦

Some comments on this definition are perhaps called for: the structure of Ψ_o does not merely mirror the join-structure of Ψ. Rather, each intermediate node in Ψ_o is generated by first computing the o-reduction of its left and right sub-trees, then joining the results, and then applying the o operation to the result of that. Consequently, a reduced evaluation tree is not in general itself an evaluation tree (though it might be).

The *raison d'être* behind this application pattern is minimality: we wish to make intermediate results as small as they can be by removing all information from a node that is redundant at that point in the evaluation tree. As it turns out, the join operator sometimes introduces new redundancies in cases where there are none in the join arguments. An example of this, studied more closely in section 7.1, is the operation of removing columns from an intermediate result once they are no longer required for joins: suppose for instance that each of two leaves share a variable $?x$. Then no reduction operator should, on pain of unsoundness, be allowed to remove that column before the join has been performed. But, $?x$ may not be involved in joins *beyond* this point, in which case it is safe to "garbage collect" it immediately *after*, calling for a second application of the operation in question.

This application pattern is also the reason why a reduction operator takes three and not two arguments: a tree, and index *and* an answer set. When a reduction operator is applied in an iterated fashion to the reduction of the left and right subtrees of a node i in a tree Ψ, then it is *not* applied to $\Omega^i \in \Psi$ but to a smaller (in the sense of Definition 6.2) set computed from it. The second case of Definition 6.1 stipulates that this computation is nevertheless determined by the index i and the original tree Ψ. We shall usually suppress the reference to the tree Ψ when it can be inferred from context, and write just o_i.

Turning now to the question of what conditions it is reasonable to place on such an operator if it is to be apt to call it a reduction operator, the following two seem to have some intuitive traction: a reduction operator should reduce the size of intermediate results, not necessarily in all cases, but results should at least never grow bigger. Secondly, a reduction operator should preserve the semantics of the final result of evaluation up to a suitable notion of equivalence. Given Definition 6.1 what this must be taken to mean is that if a size-reducing operator is applied to a tree Ψ in the iterated manner outlined by that definition, then the final result extracted (by projection, that is) from the root of the reduct Ψ_o should be the same as that extracted from the root of Ψ up to some as yet unspecified notion of equivalence.

Definition 6.2 (Reduction operation)**.** Let $\equiv$ be an equivalence relation on 2^{Σ}. A function $o : \mathbb{T} \times A^* \times \mathscr{A} \longrightarrow \mathscr{A}$ is a reduction operation wrt. $\equiv$ if it satisfies the following conditions wrt. any tree Ψ and any set of SPARQL variables W:

Result equivalence: $\pi_W(\Omega^{\epsilon}) \equiv \pi_W(\Omega^{\epsilon}_o)$

Shrinking: $s(\Omega^i_o) \leq s(\Omega^i)$ for any $i \in dom(\Psi)$, where

$$s(\Omega) =_{df} \begin{cases} |\Omega| \times |dom(\mu)| \text{ if } \Omega \neq \emptyset \text{ and } \mu \in \Omega \\ \\ s(\Omega) = 0 \text{ otherwise.} \end{cases}$$

$\circ$

The shrinking property is always entirely obvious and we usually only mention it in passing.

7. Removing superfluous columns

As illustrated by Example 7.1, the variables in a SPARQL query can be classified into three groups: 1) join variables, 2) project variables and 3) pure existence requirements.

Example 7.1. *Consider the query*

```
SELECT ?x ?y WHERE {?x :p1 ?y. ?x :p2 ?z.}
```

Its graph pattern is illustrated in Fig. 9. The variables $?x$ *and* $?y$ *are obviously project variables, but only* $?x$ *is a join variable. The variable* $?z$*, on the other hand, is neither a project variable nor a join variable. It is merely a condition that requires that there be a* $:p_2$ *edge from the value of* $?x$ *to some other entity in the data.*

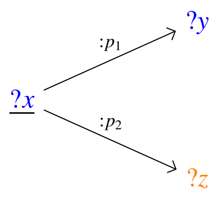
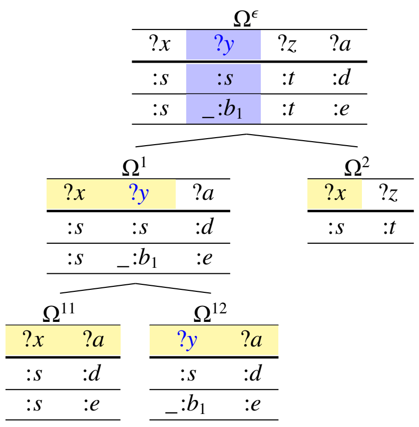


Figure 9: The different types of query variables: blue = project, orange = pure existence requirement, underlined=join.

An existence requirement's being *pure* should be taken to mean that it is not also a project variable or a join variable—after all, all variables express existence requirements. In other words, the set of pure existence requirements is disjoint from the other two. The latter two types may overlap though, since a join variable may be projected, but they are in general distinct.

The utility of this threefold classification consists in the fact that it expresses the different functions a variable may fulfill in the course of evaluating the query. This in turn provides a basis for analyzing the points at which the information that is bound to a variable is no longer necessary for computing the answer.

A pure existence condition expires, one might say, the moment it has been applied, whereas a join variable, if it not also a project variable, expires after all the partial results in which that variable occurs have been combined. A project variable never expires. Expiration points, whenever they exist, can be used to reduce the size of intermediate result.



Ω^{ϵ}

$?x$	$?y$	$?z$	$?a$
$:s$	$:s$	$:t$	$:d$
$:s$	$\_:b_1$	$:t$	$:e$

Ω^1

$?x$	$?y$	$?a$
$:s$	$:s$	$:d$
$:s$	$\_:b_1$	$:e$

Ω^2

$?x$	$?z$
$:s$	$:t$

Ω^{11}

$?x$	$?a$
$:s$	$:d$
$:s$	$:e$

Ω^{12}

$?y$	$?a$
$:s$	$:d$
$\_:b_1$	$:e$

Figure 10: Live and expired variables

Consider the evaluation tree in Fig. 10. The conventions are as follows: the project column is marked in blue. A yellow cell indicates a point at which a variable is *live*, meaning that it is semantically irredundant at that point. White cells mark *idle* positions, which are all positions occupied by a variable after it has expired.

For instance: variable $?a$ is live at Ω^{11} and Ω^{12} since the two sets join on this variable. It expires at Ω^1 after the computation of $\Omega^{11} \bowtie \Omega^{12}$ since it does not occur in any other branch of the tree. Variable $?x$, on the other hand, is live at Ω^{11} and Ω^{12} like $?a$, but it does not expire at Ω^1 since Ω^2 contains it too. The variable $?z$ is a pure existence requirement. It is not used for joins, nor is it projected. Its only purpose is to constrain the generation of solutions by expressing a condition that eligible solutions must satisfy. It therefore never occupies a live position, which means that as soon as it has been applied it is already redundant. Therefore, the corresponding columns can be removed from all intermediate results with impunity. Finally, variable $?y$ is a project variable. Since project variables represent bindings that are explicitly requested by the query, it is never superfluous and never expires.

Moving towards a formalization of these intuitive remarks, the concept of a live *join* variable is given by Definition 7.1:

Definition 7.1 (Live join variables)**.** Let Ψ be an evaluation tree and let Ω^j be any node in Ψ. Then the live join variables at j in Ψ is defined as:

$$J(\Psi, j) = dom(\Omega^j) \cap X$$

where $X = \bigcup_i dom(\Omega^i)$ for every i such that i and j are $\leq$-incomparable.

$\circ$

Applied to Fig. 10, Definition 7.1 outputs the yellow cells that are join variables. The remaining yellow cells are project variables. Hence,

Definition 7.2 (Live variables)**.** $V_i^{\Psi,W} =_{df} (W \cap dom(\Omega_i)) \cup J(\Psi, i)$. $\circ$

The superscripts on V will be omitted when clear from context. Note that pure selection constraints such as $?z$ in Fig. 10 are excluded by this definition, as they should be.

In general, an operation that removes columns from an answer set, henceforth called a *truncation operation*, can be defined in terms of projection.

Definition 7.3 (Truncation)**.** A truncation operation is a function $\theta : \mathscr{A} \longrightarrow \mathscr{A}$ satisfying the condition that $\theta(\Omega) = \pi_V(\Omega)$ for some set of variables V. $\circ$

Definition 7.2 of live variables determines a natural truncation operation.

Definition 7.4 (Live variable truncation)**.** Let Ψ be an evaluation and $\Omega^i \in \Psi$ and $W \subseteq dom(\Omega^\epsilon)$. Then $\tau_i^\Psi =_{df} \pi_{V_i}$.

$\circ$

The reader should keep in mind that the operation τ_i^Ψ, since it is defined by V_i which in turn abbreviates $V_i^{\Psi,W}$, is really parameterized by the project variables W. Mathematical rigor would require this to be explicit in the notation. However, since W is usually clear from context, or not essential to the argument, it will henceforth be left implicit.

Example 7.2 (τ is not distributive)**.** *Consider the variable $?a$ in $\Omega^1 = \Omega^{11} \bowtie \Omega^{12}$ in Fig. 10: it is in the domain of $\pi_{V_{11}}(\Omega^{11}) \bowtie \pi_{V_{12}}(\Omega^{12})$, yet it is not in the domain of $\pi_{V_1}(\Omega^1)$, since $?a$ expires at* 1. *This goes to show that that τ is not distributive. That is, there are in general evaluation trees with intermediate sets $\Omega^i = \Omega^j \bowtie \Omega^k$ such that:*

$$\tau_j^\Psi(\Omega^j) \bowtie \tau_k^\Psi(\Omega^k) \neq \tau_i^\Psi(\Omega^j \bowtie \Omega^k)$$

In these cases, the right hand side of the inequation always contains fewer variables than the left.

There are different ways to view this failure of distributivity. On an abstract level, it means that Ψ_τ is not in general an evaluation tree as we have defined that latter concept. It is not an evaluation tree because τ-reduction, as stipulated by the second case of Definition 6.1, may compress the information contained in a join into a smaller, but from a logical point of view equally information-rich package. From an essentially equivalent bottom-up perspective, one might say rather, that the failure of distributivity reflects the fact that joins can introduce redundancies: variables that are live in each of the join arguments expire in the join itself.

No matter how one prefers to think about it, the property explains why the full reduction of an evaluation tree needs to be conceptualized as in Definition 6.1 with an iteration in the recursive case.

It remains to check that everything aligns correctly, i.e. that the definition of τ and the procedure stipulated by Definition 6.1 combine to form a reduced tree with a root that is the same as the root of the original tree when truncated by the project variables. Lemma 7.1 and Lemma 7.2 give two jointly sufficient conditions for this.

Lemma 7.1 (Stability)**.** *Let Ψ be an evaluation tree and $\Omega^i = \Omega^j \bowtie \Omega^k \in \Psi$. We have $\pi_{V_i}(\pi_{V_j}(\Omega^j) \bowtie \pi_{V_k}(\Omega^k)) = \pi_{V_i}(\Omega^j \bowtie \Omega^k)$*

Proof. Let $\mu \in \pi_{V_i}(\pi_{V_j}(\Omega^j) \bowtie \pi_{V_k}(\Omega^k))$. Then $\mu = \pi_{V_i}(\pi_{V_j}(\mu_j) \cup \pi_{V_k}(\mu_k))$ for some μ_j and μ_k. Projection distributes over set union so $\pi_{V_j \cup V_k}(\mu_j \cup \mu_k) = \pi_{V_j \cup V_k}(\mu_j) \cup \pi_{V_j \cup V_k}(\mu_k)$. It is immediate from the Definition 7.1 of live join variables $(V_j \cup V_k) \cap dom(\Omega^j) = V_j$ so $\pi_{V_j \cup V_k}(\mu_j) = \pi_{V_j}(\mu_j)$ and similarly for k, and hence $\pi_{V_j \cup V_k}(\mu_j \cup \mu_k) = \pi_{V_j}(\mu_j) \cup \pi_{V_k}(\mu_k)$. It therefore suffices to show that $\pi_{V_j \cup V_k}(\mu_j \cup \mu_k) = \pi_{V_i}(\pi_{V_j \cup V_k}(\mu_j \cup \mu_k))$, which in turn only requires $J(i) \subseteq J(j) \cup J(k)$. But it is an easy consequence of Definition 7.1 that the set of live joins is antitione in the height of a node $\Omega^i \in \Psi$. Therefore, since i is above j and k it follows that $\mu = \pi_{V_i}(\mu_j \cup \mu_k)$. The converse direction is similar, so the proof is complete.

□

Lemma 7.2. *Let Ψ be a tree. Then for any $\Omega^i \in \Psi$ we have $\Omega^i_\tau = \pi_{V_i}(\Omega^i)$.*

Proof. Proof proceeds by induction on the depth of Ψ. For the base case, suppose $d(\Psi) = 0$. Then Ψ contains only one node Ω^ϵ, and that node is a leaf. By the first case of Definition 6.1 we have that $\Omega^\epsilon_\tau = \tau_\epsilon^\Psi(\Omega^\epsilon) = \pi_{V_\epsilon}(\Omega^\epsilon)$.

For the induction step, suppose that $\Omega^i = \Omega^j \bowtie \Omega^k$, and assume as induction hypothesis that $\Omega^j_\tau = \pi_{V_j}(\Omega^j)$ and $\Omega^k_\tau = \pi_{V_k}(\Omega^k)$. We want to show that $\Omega^i_\tau = \pi_{V_i}(\Omega^j \bowtie \Omega^k)$. We have

$$
\begin{aligned}
\Omega^i_\tau &= \tau^\Psi_i(\Omega^j_\tau \bowtie \Omega^k_\tau) && \text{df. } \Omega_\tau \\
&= \tau^\Psi_i(\pi_{V_j}(\Omega^j) \bowtie \pi_{V_k}(\Omega^k)) && \text{by ind. hyp.} \\
&= \pi_{V_i}(\pi_{V_j}(\Omega^j) \bowtie \pi_{V_k}(\Omega^k)) && \text{by def. of } \tau \\
&= \pi_{V_i}(\Omega^j \bowtie \Omega^k) && \text{by stability}
\end{aligned}
$$

□

The following corollary drips off immediately:

Corollary 7.3. *For any evaluation tree Ψ and any $W \subseteq dom(\Omega^\epsilon)$ we have*

$$\pi_W(\Omega^\epsilon) = \pi_W(\Omega^\epsilon_\tau)$$

Proof. Follows immediately from Lemma 7.2 and the fact that $\pi_{V_\epsilon} = \pi_W$ is an idempotent operation. □

In other words, τ satisfies the condition of result set equivalence, and since it obviously produces smaller intermediate results, it is a reduction operator wrt. equality in accordance with Definition 6.2.

Corollary 7.4. *τ is a reduction operation.*

Turning now to the size of intermediate results in Ψ_τ, are the intermediate results as small as they can be without altering the final result? As it turns out, not necessarily. For a counterexample, consider the tree in Fig. 11. Assume that W is the entire domain of Ω^ϵ. Then all variables in the leaves are live at those indices, from which it follows that the tree is τ-reduced. Nevertheless, it is easy to see that in this particular tree one can remove the column $?z$, as indicated by the shaded column, from Ω^1 without repercussions in the root—despite the fact that $?z$ is live at that point, and so will not be removed by τ.

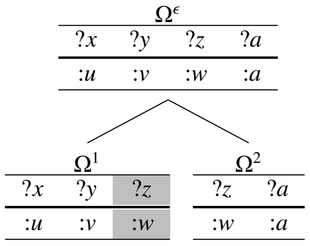


Figure 11: The possibility of reduction beyond the τ-threshold

However, it seems clear that this example, in some as yet unspecified sense, exploits the distribution of data that is specific to this particular tree. Another tree with the *same join-structure and distribution of variables*, may behave rather differently under the same operator.

Consider for instance the tree in Fig. 12. It is structurally similar to that of Fig. 11. If the $?z$ column (which occupies the same position in this tree as in that of Fig. 11) is removed from this tree, indicated by the shaded column tagged `out`, then three solution are added to ω^ϵ marked by the shaded rows

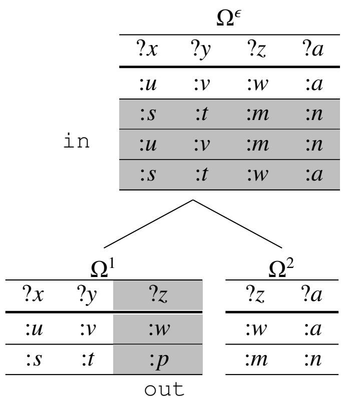


Figure 12: Same operation, non-equal root.

tagged `in`. These additional rows are incorrect or unsound solutions. Therefore, the reason why $?z$ (i.e. its absence) does not interfere with ω^ϵ in Fig. 11 must be due to the particular distribution of values in that tree.

It seems reasonable to rule out such cases, since they require the data in different branches of the tree to be scanned and compared. In a federated setting this involves shipping data from one machine to another. But of course this defeats the purpose of reducing intermediate results to begin with. It seems reasonable, therefore, to require a truncation operator to be structural in the sense that it is not sensitive to the distribution of data in the tree, but only to its join-order and distribution of variables. Formally:

Definition 7.5 (Structural truncation operation)**.** A truncation operation θ is structural iff it holds for any pair of evaluation trees Ψ and Ψ' that whenever both of the conditions below are satisfied

1. $dom(\Psi)$ and $dom(\Psi')$ are isomorphic under $\preceq$
2. $dom(\Psi(i)) = dom(\Psi'(i))$

then $dom(\theta(\Psi(i))) = dom(\theta(\Psi'(i)))$. Any pair of trees that satisfies condition (1) and (2) will be said to be *structurally similar.* ○

It is obvious that τ is structural in this sense.

Definition 7.5 suffices for a *partial* minimality result for column-reduced intermediate results: call a tree Ψ *conjunct* if every join in it is on one or more shared variables. That is, a tree is conjunct if it does not compute cartesian products. Then:

Theorem 7.5 (Column minimality)**.** *Let θ any structural truncation operation and let Ψ be a conjunct evaluation tree. Then if for any node $\Omega^k \in \Psi$ it holds that $dom(\Omega^k_\theta) \subset dom(\Omega^k_\tau)$ for some $k \in dom(\Psi)$, then θ does not satisfy result equality.*

Proof. It will be convenient to have a shorthand for talking about the properties of the relevant classes of answer sets: Henceforth a $\binom{?y \mapsto :c, :v}{:u}$-set is an answer set with two solutions $\mu^{:v}$ and $\mu^{:u}$ corresponding to each row of the expression $\binom{?y \mapsto :c, :v}{:u}$. Each row indicates a default value $:v$ and possibly a binding e.g. $?y \mapsto :c$. For instance the upper row of $\binom{?y\mapsto :c,:v}{:u}$ denotes a solution $\mu^{:v}$ that maps all variables in its domain to the same RDF constant $:v$, except $?y$ which is mapped to the RDF constant $:c$. Similarly $\binom{:v}{:u}$ denotes an answer set in which $\mu^{:v}$ maps all of its variables to $:v$ whereas $\mu^{:u}$ maps the same variables to $:u$. Finally, $\binom{-}{:u}$ denotes an answer set with only one solution $\mu^{:u}$ that maps every variable to $:u$.

Under the general assumption that the answer sets in questions are not cartesian products, i.e. that they share one or more variables, the following general join patterns are all easily verified:

1. the join of $\binom{?y\mapsto :c,:v}{:u}$-sets is a $\binom{?y\mapsto :c,:v}{:u}$-set.
2. the join of a $\binom{?y\mapsto :c,:v}{:u}$-set and a $\binom{:v}{:u}$-set is a $\binom{?y\mapsto :c,:v}{:u}$-set if $?y$ is not shared.
3. the join of a $\binom{?y\mapsto :c,:v}{:u}$-set and a $\binom{-}{:u}$-set is a $\binom{-}{:u}$-set.
4. the join of a $\binom{?y\mapsto :c_1,:v}{:u}$-set and a $\binom{?y\mapsto :c_2,:v}{:u}$-set is a $\binom{-}{:u}$-set whenever $:c_1 \neq :c_2$.
5. the join of a $\binom{-}{:u}$-set and a $\binom{-}{:u}$-set is a $\binom{-}{:u}$-set.
6. the join of a $\binom{:v}{:u}$-set and a $\binom{:v}{:u}$-set is a $\binom{:v}{:u}$-set

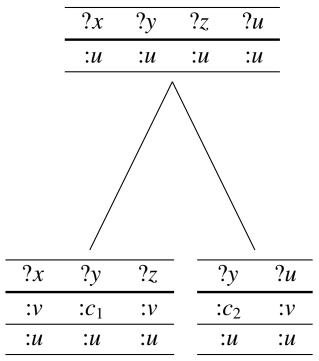


Figure 13: $\binom{?y\mapsto :c_1,:v}{:u} \bowtie \binom{?y\mapsto :c_2,:v}{:u} = \binom{-}{:u}$ whenever $:c_1 \neq :c_2$.

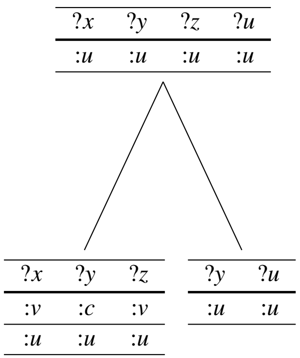


Figure 14: $\binom{?y\mapsto :c,:v}{:u} \bowtie \binom{-}{:u} = \binom{-}{:u}$.

Figs. 13 and 14 illustrate the join patterns in item 4 and 3 respectively.

Turning now to the proof proper, suppose that there is a structural truncation operator θ that yields smaller intermediate results than τ. That is, we suppose there is an evaluation tree Ψ with $\Omega^k \in \Psi$ and $?y \in dom(\Omega^k_\tau) \setminus dom(\Omega^k_\theta)$ for a structural truncation operation θ.

We need to show on the basis of this assumption that there exists an evaluation tree Ψ^* with the property that Ψ^* and Ψ^*_θ yield different result sets modulo the stipulated projection variables W. This suffices to show that θ does not satisfy result set equivalence. The proof strategy is to construct a new tree Ψ^* from Ψ that is populated with data in such a way that the final result Ω^ϵ_θ of Ψ_θ is not equal to the final result $\Omega^\epsilon_{*\theta}$ of Ψ^*_θ.

Let n be the index of the lowest join on the variable $?y$ above k in Ψ_τ, n and k are not necessarily different. That n exists follows from the assumption that $?y \in dom(\Omega^k_\tau)$, which, since $\Omega^k_\tau = \pi_{V_k}(\Omega^k)$ by Lemma 7.2 means that $?y$ is live at k and hence used in a join *above k*. The tree Ψ^* is constructed as follows.

i) $dom(\Psi^*) = dom(\Psi_\tau)$

ii) for every leaf $\Omega^{nm} \in \Psi_\tau/n1$ let $\Omega^{nm}_* \in \Psi^*$ be a $\binom{?y\mapsto :c_{n1},:v}{:u}$-set if $?y \in \Omega^{nm}_\tau$ and a $\binom{:v}{:u}$-set otherwise—in both cases with the same domain as Ω^{nm}_τ.

iii) every leaf in $\Omega^{nm} \in \Psi_\tau/n2$ determines a leaf in Ψ^* in a similar fashion to (ii), except that the binding is $?y \mapsto :c_{n2}$.

iv) for all other leaves in $\Omega^i \in \Psi_\tau$, Ω^i_* is a $\binom{?y\mapsto :c_{n2},:v}{:u}$-set if $\Omega^k_\tau \in \Psi_\tau/n2$ and a $\binom{?y\mapsto :c_{n1},:v}{:u}$-set otherwise.

Note that this construction ensures that Ψ and Ψ^* are structurally similar.

The following two observations are almost immediate: A) the root of $\Psi^*/n1$ is a $\binom{?y\mapsto :c_{n1},:v}{:u}$-set. This follows from:

- the fact that all leaves in the subtree rooted at $n1$ are either $\binom{?y\mapsto :c_{n1},:v}{:u}$-sets or $\binom{:v}{:u}$-sets by clause ii) of the construction
- join patterns 1 and 2 above, and
- the assumption that Ψ and hence Ψ^* is a conjunct tree

B) The root of $\Psi^*/n2$ is a $\binom{?y\mapsto :c_{n2},:v}{:u}$-set for the same reason as A) with the appeal to ii) replaced by an appeal to iii).

Now, from A) and B) it follows in turn that the root of Ψ^*/n is a $\binom{-}{:u}$-set, by one appeal to join pattern 4. Thus, since $\binom{-}{:u}$-sets act as zeros for conjunct answer sets, by list item 3 and 5, $\binom{-}{:u}$ propagates to the top of Ψ^* making Ω^ϵ_* a $\binom{-}{:u}$-set too. Since τ is a reduction operation we have that $\Omega^\epsilon_{*\theta}$ is a $\binom{-}{:u}$-set, so the proof is now reduced to showing that $\Omega^\epsilon_{*\theta}$ is *not* a $\binom{-}{:u}$-set.

The trees Ψ^* and Ψ are structurally similar by construction. By the main supposition of the theorem $?y \notin dom(\Omega^k_\theta)$. But θ is structural so $dom(\Omega^k_\theta) = dom(\Omega^k_{*\theta})$ and hence $?y \notin dom(\Omega^k_{*\theta})$. Appealing to structure once more we have that Ω^n_* is the lowest join on $?y$ above k in Ψ^*_θ, which means that $?y$ once it is removed from $\Omega^k_{*\theta}$ does not occur anywhere between k and n in Ψ^*_θ, in particular it does not occur in $\Omega^{n1}_{*\theta}$. It follows that $\Omega^{n1}_{*\theta}$ is a $\binom{:v}{:u}$-set.

Now, depending on whether or not $?y$ is in the domain of $\Psi^*_\theta/n2$ (we are not allowed to assume that θ removes $?y$ *only* at index k) $\Omega^{n2}_{*\theta}$ is either a $\binom{?y\mapsto :c_{n2}, :v}{:u}$-set or a $\binom{:v}{:u}$-set. In either case, if $\Omega^n_{*\theta}$ is the root of Ψ^*_θ then we are done. This follows from the fact that the root of Ψ^* is a $\binom{-}{:u}$-set, and as such cannot be made equal to a $\binom{:v}{:u}$-set by projection onto *any* set of variables W. If, on the other hand, $\Omega^n_{*\theta}$ is *not* the root of Ψ^*_θ, then it has a sibling $\Omega^m_{*\theta}$. By (*iv*) above, all leaves under m in Ψ^* are $\binom{?y\mapsto :c_{n2}, :v}{:u}$-sets, so depending on whether or not $?y$ is in both of $\Omega^{m1}_{*\theta}$ and $\Omega^{m2}_{*\theta}$ (again we are not allowed to assume that θ removes $?y$ *only* at index k) $\Omega^{m2}_{*\theta}$ is either a $\binom{?y\mapsto :c_{n2}, :v}{:u}$-set or a $\binom{:v}{:u}$-set. By list items 2 and 6 above, a $\binom{:v}{:u}$-set acts as a unit. Hence, the property of being either a $\binom{?y\mapsto :c_{n2}, :v}{:u}$-set or a $\binom{:v}{:u}$-set propagates to the root of Ψ^*_θ, yielding the desired conclusion that θ does not preserve result equality. The essence of this argument is captured by Figs. 15 and 16 which schematize the effect that θ's omission of the join variable $?y$ in Ψ^* has on the final result of Ψ^*_θ on a tree in the the case that Ψ^* is of this particular form.

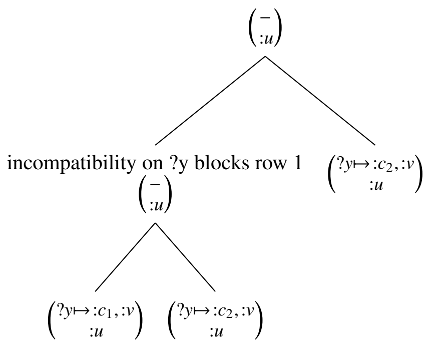

$\binom{-}{:u}$
incompatibility on ?y blocks row 1
$\binom{-}{:u}$
$\binom{?y\mapsto :c_2, :v}{:u}$
$\binom{?y\mapsto :c_1, :v}{:u}$
$\binom{?y\mapsto :c_2, :v}{:u}$

Figure 15: A sample tree Ψ^*

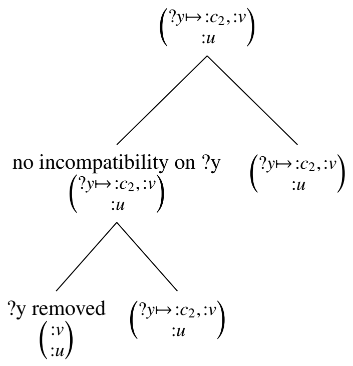

$\binom{?y\mapsto :c_2, :v}{:u}$
no incompatibility on ?y
$\binom{?y\mapsto :c_2, :v}{:u}$
$\binom{?y\mapsto :c_2, :v}{:u}$
?y removed
$\binom{:v}{:u}$
$\binom{?y\mapsto :c_2, :v}{:u}$

Figure 16: The tree Ψ^*_θ

It remains to prove the case where $?y \in W$ but $?y$ is not a join variable. But this case is trivial since it implies that $?y$ occurs in just one leaf, so once it is removed it will not reappear higher up in the tree. Therefore, there are no bindings for $?y$ in $\pi_W(\Omega^\epsilon_{*\theta})$ making it different from $\pi_W(\Omega^\epsilon_*)$.

□

Taking stock, the initial question about the size of intermediate results in Ψ_τ has been partially answered: if Ψ is conjunct, then the intermediate results in Ψ_τ are as small as any *structural* operation can make them. Whether the restriction to conjunct trees can be removed is not clear at the time of writing, and is left as an open question.

Excursus. Some connections to related work. The general idea of removing columns in intermediate results for certain purposes and operations is not one we claim originality for. Similar ideas have a long pedigree in query optimization. The history of the topic is rather tortuous, and the web of interconnections too intricate to do it justice here. Suffice it for present purposes to make a couple of observations.

The *semijoin* strategy deserves mention since it was designed for reducing the number of columns shipping overhead in the case where data is distributed among different sites. The use of semijoins for distributed databases was first introduced in [24] and [25]. Here, we explain the said strategy in terms of the nomenclature and examples of the present paper: Applied to the intermediate nodes Ω^1 and Ω^2 in Fig. 10 the semijoin strategy is (cf. [26, p. 856]):

1. compute $m := \pi_{dom(\Omega^1)\cap dom(\Omega^2)}(\Omega^1)$ at node 1
2. ship m from node 1 to node 2
3. compute $n := \Omega^2 \bowtie s$ at node 2
4. ship n from node 2 to node 1
5. compute $\Omega^1 \bowtie n$ at node 1. This is the same as $\Omega^1 \bowtie \Omega^2$

The third step yields an intermediate result that is the same as Ω^2 with the column corresponding to the pure existence constraint $?z$ removed. To remove $?z$ is also what τ does, so the semijoin strategy and τ agree on the redundancy of $?z$.

However the differences are fairly obvious. First, as witnessed by step 5 above, the semijoin strategy passes the result of the join up the tree in its entirety, no columns are actually removed. Truncation, in contrast, deletes a column from its expiration point onwards.

More substantially, the two strategies do not have the same operational semantics. The semijoin strategy has a local scope that looks only at the arguments being joined, whereas truncation has a global scope that also considers occurrences of variables in other branches of the tree.

Although, the semijoin procedure and the τ operation agree on the superfluousness of $?z$ in Ω^2 in 10, the semijoin strategy looks only at Ω^1 and Ω^2 to determine this, whereas the truncation operator takes into considerations all other positions in the tree. This difference becomes clearly visible when we elaborate on the example by having $?z$ occur in another branch further up, as in Fig. 17. In this tree $?z$ is a live variable by our definition, whereas it is still idle in the semi-join strategy *at that point*. Incidentally, this example also goes to show that the concept of a

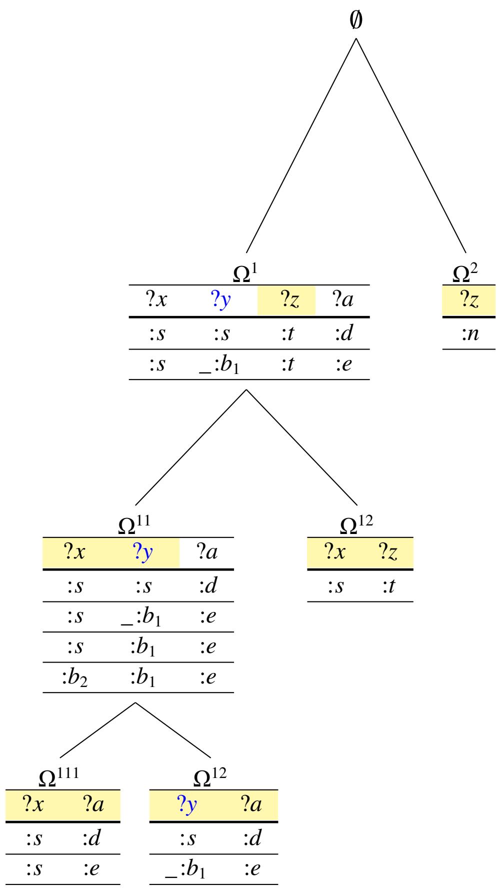


Figure 17: Semi-joins compared with live variables

pure existence requirement—$?z$ is a pure existence requirement in Fig. 10—is global in the same sense as live variables. It therefore differs from semi-joins for the same reason.

The truncation operator τ is rather more closely related to *projection pushing* as developed in (among others) [27, 28, 29]. The idea behind this strategy is to "push" projections as far down the evaluation tree as possible. This generally reduces the size of intermediate results [30]. The process must of course preserve join variables on its way down lest constraints on the combination of intermediate results be violated. Projection pushing is therefore sometimes defined recursively, starting from an initial set of projection variables and the root of a tree, and then gradually expanding the set of variables as it is pushed to include the necessary joins at each level. One such recursive formulation appears in [6] in the context of RDF and SPARQL. It takes the form of a set of rewriting rules for distributing projection expressions over joins. The PJPush rule reads as follows (adapted to the present nomenclature):

$$\pi_W(\Omega^i \bowtie \Omega^j) = \pi_W(\pi_{W'}(\Omega^i) \bowtie \pi_{W'}(\Omega^j)) \qquad (2)$$

where W' is defined as $W \cup (var(\Omega^i) \cap var(\Omega^j))$. It is easy to check that when W is passed from the top of the tree and down, then this rule computes exactly the variables that are live according to Definition 7.2.

When in the present paper we have opted for a non-recursive operator definition, it is for two related reasons: first it allows us to display projection pushing as a particular instance of an abstract pattern shared by all reduction operations. Secondly, this shared abstract stratum allows the interaction between reduction operators to be studied. It is a question of forming new and complex reduction operations by composition of elementary ones. To repeat from the introduction, one of the main findings of the present paper is that when row-reduction and column-reduction operations are made to act in consort—and are applied *in the right order*—they reduce intermediate results beyond the threshold of the row operation taken in isolation. That is, the reduct of an answer set under a certain complex operation will contain fewer rows than the reduct of the same set under the row operation that is a factor of the complex one. The demonstration of this requires a uniform formalization.

8. Removing rows with blank nodes in join position

Recall the idle join assumption from section 5: a blank node in join position can not be, and never needs to be, combined with any solution from any other partial answer unless they are on the same path. By the completeness result of [3], this is true in general for any federated evaluation tree that directs different portions of a query to different sites.

There is an obvious heuristic opportunity in this: for any solution in any intermediate result, if it has a blank node in join position, then it is superfluous. This is the simple idea developed in the present section. We show that the operation taking an answer set to a smaller one where the mentioned rows are removed is a reduction operation in the sense of Section 6.

Definition 8.1. Let Ψ be an evaluation tree, $i \in dom(\Psi)$, and let Ω be any answer set. Then

$$\sigma_i^{\Psi}(\Omega) =_{df} \{\mu \in \Omega \colon \forall ?x \in J(\Psi, i) \cap dom(\mu) \rightarrow \mu(?x) \notin B\}$$

○

In words, given an evaluation tree Ψ and a node index i, $\sigma_i^{\Psi}(\Omega)$ computes the subset of Ω where all elements have concrete values for the join variables that are live at point i in the evaluation tree.

Example 8.1. *Consider the evaluation tree in Fig. 18. The live join positions have been marked in yellow. Fig. 19 shows the result of applying the operation σ to nodes in the tree with the corresponding index. All changes flow from removing the second row from Ω^{11}. This row has a blank node for the variable $?x$, and $?x$ is a live join at this point. It follows that the row in question is not compatible with any other solution elsewhere in the tree, and therefore that it is semantically redundant. Consequently, removing it does not change the semantics of the final result.*

We will show that σ is a reduction operation wrt. equality. Obviously σ reduces the size of intermediate results, so we move on from shrinking to result-equivalence. We begin by stating an obvious, but important property regarding the sharing

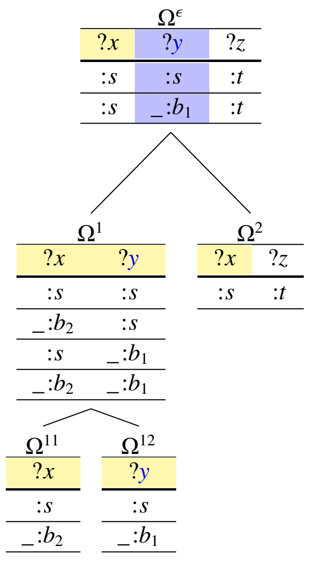


Figure 18: An evaluation tree.

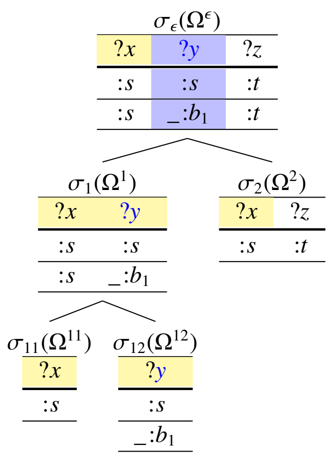


Figure 19: the σ-reduction of Fig. 18.

of blank nodes in intermediate results in a distributed evaluation tree. This is the disjointness property for blank nodes that was alluded to in Section 5:

Lemma 8.1. *If $\Omega^j, \Omega^k \in \Psi$, $j \not\preceq k$ and $j \not\succeq k$, then $ran(\Omega^j) \cap ran(\Omega^k) \cap B = \emptyset$*

Proof. For contradiction, assume $b \in ran(\Omega^m) \cap ran(\Omega^n) \cap B$. Since for any pair of leaves $\Omega^m, \Omega^n \in l(\Psi)$ where $m \neq n$ then $ran(\Omega^m) \cap ran(\Omega^n) \cap B = \emptyset$, it must be that there is a leaf $\Omega^i \in l(\Psi)$ s.t. $j \preceq i$ and $k \preceq i$. But then j and k are both string prefixes of i, hence it must be that either $j \preceq k$ or $j \succeq k$, either way a contradiction. □

The lemma says that no sibling branches share blank nodes, a direct consequence of leaf nodes not sharing blank nodes and the tree-structure. Contrapositively, two intermediate results share blank nodes only if they both lie on the same branch in the tree.

Returning focus to σ, Lemma 8.2 establishes that applying σ_i^Ψ to the node $\Omega^i \in \Psi$ for which it was defined from, produces the node Ω^i_σ in the reduced tree Ψ_σ—it is analogous to Lemma 7.2.

Lemma 8.2. *Let Ψ be a tree. Then for any $\Omega^i \in \Psi$ we have*

$$\Omega^i_\sigma = \sigma_i^\Psi(\Omega^i)$$

Proof. Proof proceeds by induction on the depth of Ψ.

Base case: $d(\Psi) = 0$. Then Ψ contains only one node Ω^ϵ, and that node is a leaf. By the first case of Definition 6.1 of a reduced tree we have that $\Omega^\epsilon_\sigma = \sigma_\epsilon^\Psi(\Omega^\epsilon)$.

Induction step: $d(\Psi) > 0$. Suppose that $\Omega^i = \Omega^j \bowtie \Omega^k$. We need to show that $(\Omega^j \bowtie \Omega^k)_\sigma = \sigma_i^\Psi(\Omega^j \bowtie \Omega^k)$. By the second case of Definition 6.1 we have that $(\Omega^j \bowtie \Omega^k)_\sigma = \sigma_i^\Psi(\Omega^j_\sigma \bowtie \Omega^k_\sigma)$ so it suffices to show that $\sigma_i^\Psi(\Omega^j \bowtie \Omega^k) = \sigma_i^\Psi(\Omega^j_\sigma \bowtie \Omega^k_\sigma)$. Assume as induction hypothesis (IH) that $\Omega^j_\sigma = \sigma_j^\Psi(\Omega^j)$ and $\Omega^k_\sigma = \sigma_k^\Psi(\Omega^k)$. The proof then reduces to showing $\sigma_i^\Psi(\Omega^j \bowtie \Omega^k) = \sigma_i^\Psi(\sigma_j^\Psi(\Omega^j) \bowtie \sigma_k^\Psi(\Omega^k))$ We prove each inclusion separately.

($\Rightarrow$). Suppose that $\mu \in \sigma_i^\Psi(\Omega^j \bowtie \Omega^k)$ for $\mu = \mu_j \cup \mu_k$ such that $\mu_j \in \Omega^j$ and $\mu_k \in \Omega^k$. Assume for contradiction that $\mu \notin \sigma_i^\Psi(\sigma_j^\Psi(\Omega^j) \bowtie \sigma_k^\Psi(\Omega^k))$. Since $\sigma_j^\Psi(\Omega^j) \bowtie \sigma_k^\Psi(\Omega^k) \subseteq \Omega^j \bowtie \Omega^k$ and, by the first assumption, $\mu \in \Omega^j \bowtie \Omega^k$, it follows by the second assumption that $\mu_j \cup \mu_k \notin \sigma_j^\Psi(\Omega^j) \bowtie \sigma_k^\Psi(\Omega^k)$. Hence, by the definition of $\bowtie$ either $\mu_j \notin \sigma_j^\Psi(\Omega^j)$ or $\mu_k \notin \sigma_k^\Psi(\Omega^k)$. The two cases are similar, so assume wlog. that the former is the case. Then since $\mu_j \in \Omega^j$ it follows by Definition 8.1 that there is an $?x \in dom(\Omega^j)$ such that $?x \in J(\Psi, j)$ and $\mu_j(?x) \in B$. Now, since $?x \in J(\Psi, j) \backslash J(\Psi, i)$, it follows that $?x$ expires at i whence $?x \in J(\Psi, k)$. Since $\mu_j \leftrightharpoons \mu_k$ and $?x \in dom(\mu_j) \cap dom(\mu_k)$ it must be that $(\mu_j \cup \mu_k)(?x) = \mu_j(?x) = \mu_k(?x)$. Therefore, since $?x \in J(\Psi, j) \cup J(\Psi, k)$ and $\mu_j \cup \mu_k \notin \sigma_j^\Psi(\Omega^j) \bowtie \sigma_k^\Psi(\Omega^k)$, we may infer that $\mu_j \cup \mu_k(?x) = \mu_j(?x) = \mu_k(?x) \in B$. But this contradicts Lemma 8.1 which says that $ran(\Omega^j) \cap ran(\Omega^k) \cap B = \emptyset$.

($\Leftarrow$). This follows directly from the fact that $\sigma_j^\Psi(\Omega^j) \bowtie \sigma_k^\Psi(\Omega^k) \subseteq \Omega^j \bowtie \Omega^k$ and the monotony of σ_i^Ψ.

□

Proving result-set equivalence in the sense of Definition 6.2 is now fairly straightforward. Here as in the previous section result-set equivalence is actually result set *equality*.

Theorem 8.3. *For any evaluation tree Ψ and any $W \subseteq dom(\Omega^\epsilon)$*

$$\pi_W(\Omega^\epsilon) = \pi_W(\Omega^\epsilon_\sigma)$$

Proof. By Lemma 8.2 we have $\Omega^\epsilon_\sigma = \sigma^\Psi_\epsilon(\Omega^\epsilon)$, so it suffices to show that $\Omega^\epsilon = \sigma^\Psi_\epsilon(\Omega^\epsilon)$ because then $\Omega^\epsilon = \Omega^\epsilon_\sigma$ so $\pi_W(\Omega^\epsilon) = \pi_W(\Omega^\epsilon_\sigma)$ as desired. Now, Ω^ϵ is the root node of Ψ, so $J(\Psi, \epsilon) = dom(\Omega^\epsilon) \cap (\Psi \setminus (\Psi/\epsilon)) = dom(\Omega^\epsilon) \cap (\Psi \setminus \Psi) = dom(\Omega^\epsilon) \cap \emptyset = \emptyset$. It follows that the condition $\forall\, ?x \in J(\Psi, \epsilon) \rightarrow \mu(?x) \notin B$ is vacuously true for any $\mu \in \Omega^\epsilon$ so $\Omega^\epsilon \subseteq \sigma^\Psi_\epsilon(\Omega^\epsilon)$. Since the converse inclusion holds trivially, this completes the proof. □

Again, it is entirely evident that σ produces smaller intermediate results, i.e. that it satisfies the reduction property of Definition 6.2. In conclusion we have:

Corollary 8.4. *σ is a reduction operation.*

In summary, σ-reduction utilizes the special case where we know that a certain type of values, that of blank nodes, will never be able to join with other results. In other words, it doesn't require peeking ahead at other intermediate results or any such prescient behavior. It does this in a way that preserves answers at the root, as it should.

It is worth noting that, in contrast to τ-reduction (cf. Example 7.2), σ-reduction yields an evaluation tree. That is, the tree Ψ_σ, obtained by reducing evaluation tree Ψ by σ, satisfies $\Psi_\sigma(n1) \bowtie \Psi_\sigma(n2) = \Psi_\sigma(n)$. This is ultimately due to the distributivity of σ, which is established in the following lemma:

Lemma 8.5. *Let Ψ be a tree. Then for any node $\Omega^i \in \Psi$ of the form $\Omega^i := \Omega^j \bowtie \Omega^k \in \Psi$, then*

$$\sigma^\Psi_i(\Omega^j \bowtie \Omega^k) = \sigma^\Psi_j(\Omega^j) \bowtie \sigma^\Psi_k(\Omega^k)$$

Proof. by Lemma 8.2, we have that $\sigma^\Psi_i(\Omega^j \bowtie \Omega^k) = (\Omega^j \bowtie \Omega^k)_\sigma$, $\sigma^\Psi_j(\Omega^j) = \Omega^j_\sigma$ and $\sigma^\Psi_k(\Omega^k) = \Omega^k_\sigma$. By the second case of Definition 6.1, we have that $(\Omega^j \bowtie \Omega^k)_\sigma = \sigma^\Psi_i(\Omega^j_\sigma \bowtie \Omega^k_\sigma)$, hence $\sigma^\Psi_i(\Omega^j \bowtie \Omega^k) = \sigma^\Psi_i(\Omega^j_\sigma \bowtie \Omega^k_\sigma)$. Since $\Omega^j_\sigma \bowtie \Omega^k_\sigma = \sigma^\Psi_j(\Omega^j) \bowtie \sigma^\Psi_k(\Omega^k)$, then by equality, and functionality of σ^Ψ_i, $\sigma^\Psi_i(\Omega^j \bowtie \Omega^k) = \sigma^\Psi_i(\sigma^\Psi_j(\Omega^j) \bowtie \sigma^\Psi_k(\Omega^k))$, hence it suffices to show $\sigma^\Psi_i(\sigma^\Psi_j(\Omega^j) \bowtie \sigma^\Psi_k(\Omega^k)) = \sigma^\Psi_j(\Omega^j) \bowtie \sigma^\Psi_k(\Omega^k)$. We show this by set inclusion. Since $\sigma^\Psi_i(\sigma^\Psi_j(\Omega^j) \bowtie \sigma^\Psi_k(\Omega^k)) \subseteq \sigma^\Psi_j(\Omega^j) \bowtie \sigma^\Psi_k(\Omega^k)$, it remains to show $\sigma^\Psi_i(\sigma^\Psi_j(\Omega^j) \bowtie \sigma^\Psi_k(\Omega^k)) \supseteq \sigma^\Psi_j(\Omega^j) \bowtie \sigma^\Psi_k(\Omega^k)$. Suppose that $\mu \in \sigma^\Psi_j(\Omega^j) \bowtie \sigma^\Psi_k(\Omega^k)$ for $\mu = \mu_j \cup \mu_k$ such that $\mu_j \in \sigma^\Psi_j(\Omega^j)$ and $\mu_k \in \sigma^\Psi_k(\Omega^k)$. Assume for contradiction that $\mu \notin \sigma^\Psi_i(\sigma^\Psi_j(\Omega^j) \bowtie \sigma^\Psi_k(\Omega^k))$. That is, by definition, there is an $?x \in dom(\mu_j \cup \mu_k) \cap J(\Psi, i)$ s.t. $\mu_j \cup \mu_j(?x) \in B$.

We break the proof down into cases, based on the membership of $?x$ in the domains of μ_j and μ_k.

Case: $?x \in dom(\mu_j) \cap dom(\mu_k)$. Then, since $\mu_j \leftrightharpoons \mu_k$, it follows that $\mu_j(?x) = \mu_k(?x) \in B$. By Lemma 8.1, $ran(\Omega^j) \cap ran(\Omega^k) \cap B = \emptyset$, hence it follows that $ran(\mu_j) \cap ran(\mu_k) \cap B = \emptyset$, hence $\mu_j(?x) = \mu_k(?x) \notin B$, a contradiction.

Case: $?x \notin dom(\mu_j) \cap dom(\mu_k)$. That is, $?x \in dom(\mu_j)$ or $?x \in dom(\mu_k)$, but not in both. Since two are similar, then wlog. assume $?x \in dom(\mu_j)$. Since $J(\Psi, i) \subseteq J(\Psi, j) \cup J(\Psi, k)$, then $?x \in dom(\mu_j) \cap (J(\Psi, j) \cup J(\Psi, k))$. Now, since $dom(\Omega^j) \cap J(\Psi, k) \setminus J(\Psi, j) = \emptyset$ as a consequence of Definition 7.1, then $?x \in dom(\mu_j) \cap J(\Psi, j)$ and $\mu_j(?x) \in B$. But since $\mu_j \in \sigma^\Psi_j(\Omega^j)$ then $\mu_j(?x) \notin B$, a contradiction.

□

Distributivity may also be viewed as a *leaf-sufficiency* principle, meaning that *all* intermediate results can be minimized by just reducing the leaves. It is a property that the σ-operation is the sole operation in this paper to enjoy. This difference between the operators has the consequence, among other things, that a σ-reduced tree and a, say, τ-reduced one are not of the same form. Therefore, if one wants to compose these operators, one has to take special steps to make sure that they type check. We return to this topic in Section 10.

9. Information preserving subsets

There is another property that one may hope to exploit in order to reduce the number of rows in an intermediate result without compromising the logical content of the final answer, namely the relative *informativeness of solutions*. For the purposes of this informal discussion, let rows in an answer set be denoted by tuples. Then it is intuitively clear that the row $(:s, :s)$ is more specific than $(:s, \_:b)$. Therefore, an answer set that contains only the former may plausibly be said to contain just as much information (under set semantics) as an answer set that contains both.

For illustration, consider the two answer sets in Fig. 20. It seems intuitively clear that the top row in the table on the left is more informative than the bottom one, and therefore that the table on the right contains no less information than that on the left. Conversely, since the table on the right is subsumed by that on the left, the left also entails the right, so the two sets are intuitively equivalent under set semantics. In other words, the row containing a blank node can in this case be deleted. To be sure, in more complex cases, cross-references between blank nodes in different rows may need to be taken into account.

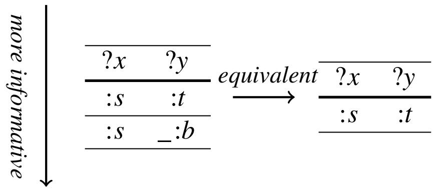


Figure 20: Intuitively equivalent sets.

It is the purpose of the present section to make these intuitive remarks formally precise. This requires that the concepts of an answer set 'preserving' the information of another answer set be defined and justified in a meaningful and plausible way.

The following roadmap to the present section might be helpful: First we define information preservation in terms of the existence of a valuation function between answer sets, and use this notion to coin a row-reduction operation. A valuation function instantiates blank nodes in answer sets in a way similar to how nulls are treated in the theory of incomplete databases [4], except that valuations are here generalized to allow mappings from blank nodes to other blank nodes. In this respect valuations are rather more like RDF homomorphisms.

Next, we generalize the concept of RDF-entailment to that of SPARQL-entailment. The latter is defined as a homomorphic functional relation that holds between graph *patterns*, and induces a concept of SPARQL-*equivalence* as the two-way entailment between such patterns. A concept of SPARQL equivalence is needed because, unlike the reduction operators discussed heretofore, the operation of removing less informative rows from intermediate results is obviously not a reduction operator wrt. *equality* (ref. Definition 6.2), witnessed by Fig. 20. We shall need a less stringent concept of equivalence therefore.

Finally, we show that the operation of removing less informative solutions from an intermediate result preserves the final result of query processing up to SPARQL-equivalence. That is, reducing intermediate answer sets by way of valuations will preserve final results in a sense that conforms to the abstract Definition 6.2 of a reduction operator.

Before bringing the section to a close, we also take the time to compare our concept of result set equivalence with the concept of *completions* from the theory of incomplete databases. As it turns out, there is an instructive relationship between the two under the open world semantics, so called in [4]. This relationship reflects a duality between information preserving reductions, on the one hand, and the addition of tuples on the other hand, in this case redundant ones, that is permitted by the open world semantics.

9.1. SPARQL-equivalence

Recall the following concepts from [19] (which derives ultimately from [31]).

Definition 9.1 (RDF homomorphism)**.** An RDF homomorphism $h : G_1 \to G_2$ is a function from IBL to IBL such that $h(u) = u$ for every $u \in IL$ and such that $(u_1, u_2, u_3) \in G_1$ implies $(h(u_1), h(u_2), h(u_3)) \in G_2$. ◦

Definition 9.2 (RDF-entailment)**.** An RDF graph G_1 entails another G_2 iff there is an RDF homomorphism h from G_2 to G_1. ◦

The problem with these definitions, for the purposes of the present paper, is that since they hark back to RDF homomorphisms they only apply to pairs of answer sets Ω and patterns P with the property that the variables that occur in P are all in the domain of Ω. If this condition is not satisfied then $\mu(P)$, for any $\mu \in \Omega$, is not an RDF graph and cannot therefore be the source or the target for an RDF homomorphism.

This limitation is important because the projection operation that produces the final answer $\pi_W(\Omega^\epsilon)$ from the root Ω^ϵ of a tree Ψ will not in general contain all the variables that occur in the query pattern. That is, even though $var(P) \subseteq dom(\Omega^\epsilon)$ holds in general $var(P') \subseteq dom(\pi_W(\Omega^\epsilon))$ may *not* hold for any subset P' of P. It follows that $\pi_W(\Omega^\epsilon)$ and Ω^ϵ will not in general be RDF-equivalent, and therefore that the operation of removing less informative rows will not be a reduction operation wrt. RDF-equivalence in the sense of Definition 6.2.

We therefore propose the following generalization of RDF homomorphisms to *SPARQL homomorphisms*:

Definition 9.3 (SPARQL homomorphism)**.** A SPARQL homomorphism is a function from $IBL\mathcal{V}$ to $IBL\mathcal{V}$ that maps a BGP P_1 homomorphically to a BGP P_2 under the condition that $h(u) = u$ for $u \in IL\mathcal{V}$. ◦

In analogy to the relation between RDF-homomorphisms and RDF-entailment Definition 9.3 induces a concept of SPARQL-entailment:

Definition 9.4 (SPARQL entailment)**.** Let P_1, P_2 be BGPs. Then $P_1 \vdash P_2$ iff there is a SPARQL-homomorphism from P_2 to P_1. ◦

Answer set-equivalence can now be defined as follows:

Definition 9.5 (Answer set entailment)**.** Let Ω^i and Ω^j be answer sets with the same domain and P a basic graph pattern. We shall say that Ω^i SPARQL-entails Ω^j modulo P, written $\Omega^i \vdash_P \Omega^j$, iff

$$\bigcup_{\mu \in \Omega^i} \mu(P) \vdash \bigcup_{\mu' \in \Omega^j} \mu'(P)$$

We let $[\Omega]_{\dashv\vdash_P} \subseteq \mathscr{A}$ denote the equivalence class induced by $\dashv\vdash_P$. ◦

Taking stock so far, one might say that answer set equivalence based on SPARQL-entailment—henceforth referred to simply as answer set entailment—is what one gets if one blurs the difference between a variable and a constant and considers a BGP as itself a kind of RDF graph. Mapping *queries* onto one another in this manner is a well-known strategy in database theory [30].

9.2. Informativeness and information preservation

We start by defining information preserving *valuations* of blank nodes.

Definition 9.6. A valuation is a function $v : B \to IBL$. We shall say that an answer set Ω^j preserves the information in Ω^i iff there is a v such that $v(\Omega^i) = \Omega^j$. ◦

This concept is a blend of a valuation as found in database literature (e.g. in [4]), and the concept of SPARQL homomorphisms. More on this shortly.

Information preserving valuations of nulls, the database equivalent to blank nodes, is well studied (see [4, 5, 30]). We give a quick recap of the central notions from [4, 5] to throw our own concept into relief:

Incomplete databases have two kinds of values: *constants* from an infinite set `Const` and *nulls* from an infinite set `Null`. A relational *schema* is a set of relation names with associated arities. A *database instance* `D` assigns to each k-ary relation `R` from the schema a k-tuple over `Const` ∪ `Null`. Set of constants and nulls that occur in `D` are denoted `Const(D)` and `Null(D)` respectively. A database instance is *complete* if every tuple has values from `Const` and *incomplete* otherwise. A *valuation* is a function v : `Null(D)` → `Const` that completes an incomplete database by uniformly substituting constants for nulls. The completion of `D` by v is denoted v(`D`). The semantic denotation of an incomplete database D is defined as the set of its completions, which can be taken in an *open world*, sense and a *closed world* sense:

Open world semantics

$$[\![D]\!]_{OWA} =_{def} \left\{ D' \;\middle|\; \begin{array}{l} D' \text{ is complete and} \\ v(D) \subseteq D' \text{ for some valuation } v \end{array} \right\}$$

Closed word semantics

$$[\![D]\!]_{CWA} =_{def} \left\{ D' \;\middle|\; D' = v(D) \text{ for some valuation } v \right\}$$

A database D′ is said to be *more informative* than D if D′ $\in [\![\mathtt{D}]\!]_*$ for $* \in \{OWA, CWA\}$.

The difference between this database concept of a valuation and that of Definition 9.6 is minor but of consequence: valuations in the present sense are allowed to map blank nodes not only to constants but also to literals and more importantly to other blanks. Example 9.1 shows the behavior of valuations on answer sets.

Example 9.1. *Consider the query $P = \{(?x, :p, ?y)\}$ and the following answer set*

$\Omega =$

$?x$	$?y$
$:s$	$:t$
$:s$	$\_:b_1$
$:s$	$\_:b_2$
$\_:b_2$	$:t$

Let v^i be the valuation that maps all blank nodes to $:t$. Then

$v^i(\Omega) =$

$?x$	$?y$
$:s$	$:t$
$:t$	$:t$

The result in this case, is that $v^i(\Omega)$ preserves the information in Ω in that $v^i(\Omega) \vdash_P \Omega$. Note, however, that equivalence does not hold since the edge $(:t, :p, :t)$ in $(v^i(\Omega))(P)$ is not contained in $\Omega(P)$, hence $v^i(\Omega) \ndashv_P \Omega$.

Now, if we let v^j be the valuation that maps $\_:b_1 \mapsto :t$ and identity for all other blank nodes. Then

$v^j(\Omega) =$

$?x$	$?y$
$:s$	$:t$
$:s$	$\_:b_2$
$\_:b_2$	$:t$

In this case, however, we do have equivalence in the form that $v^i(\Omega) \dashv\vdash_P \Omega$

As shown in the abovementioned example, valuations act more like SPARQL- or RDF homomorphisms, which will turn out to be what will be required for preserving answer set equivalence. This is not too surprising since by Definition 9.5 the latter concept is a relation that holds between *graph* patterns. The relation of being a reduct of an answer set modulo such a valuation can then straightforwardly be defined as the former's being an equally informative subset of the latter. Lemma 9.1 addresses the first part of this programme, showing that valuations in the present sense outputs SPARQL entailed answer sets.

Lemma 9.1. *If $v(\Omega^i) = \Omega^j$ then $\Omega^j \vdash_P \Omega^i$.*

Proof. It suffices to show that there is a SPARQL homomorphism from $\Omega^i(P)$ to $\Omega^j(P)$ on the assumption that $v(\Omega^i) = \Omega^j$. Let $v^+ = v \cup \{(u, u) \mid u \in IL\}$ and suppose $t' \in \Omega^i(P)$. Then there is a $t \in P$ and a $\mu \in \Omega^i$ such that $t' = \mu(t)$. Since $v(\Omega^i) = \Omega^j$ it follows that $v\mu \in \Omega^j$, and, since v^+ is the identity on everything except possibly blank nodes, also $v^+\mu \in \Omega^j$. Therefore $v^+\mu(t) \in \Omega^j(P)$ so v^+ is a SPARQL homomorphism. □

It is worthwhile to note that the converse of Lemma 9.1 does not hold, i.e. being-a-valuation-of and being-entailed is not the same concept, viz. Example 9.2:

Example 9.2. *Consider $P = \{(?x_1, :p, ?y_1), (?x_2, :p, ?y_2)\}$ and the following two answer sets*

$\Omega^i =$

$?x_1$	$?y_1$	$?x_2$	$?y_2$
$:s$	$:t$	$:u$	$:v$

$\Omega^j =$

$?x_1$	$?y_1$	$?x_2$	$?y_2$
$:u$	$:v$	$:s$	$:t$

Then $\Omega^i(P) \vdash \Omega^j(P)$ and $\Omega^i(P) \dashv \Omega^j(P)$, but there are no valuations v^i, v^j that can make $v^i(\Omega^i) = \Omega^j$ resp. $v^j(\Omega^j) = \Omega^i$, since this would require v^i to map $:s \mapsto :u$ and $:t \mapsto :v$, and the reverse for v^j, breaking the condition that concrete terms only map to identity.

Turning to the second part of the aforementioned programme, Lemma 9.2 establishes that if we restrict the set of valuations to those that yield subsets, we end up with *equivalent* answer sets *modulo an arbitrary P*.

Lemma 9.2. *If $v(\Omega^i) = \Omega^j$ and $\Omega^j \subseteq \Omega^i$, then $\Omega^j \in [\Omega^i]_{\dashv\vdash_P}$ for any query pattern P.*

Proof. Immediately since $\Omega^i \subseteq \Omega^j$ implies $\Omega^j \vdash_P \Omega^i$, and from Lemma 9.1, we have that $v(\Omega^j) = \Omega^i$ implies $\Omega^i \vdash_P \Omega^j$.

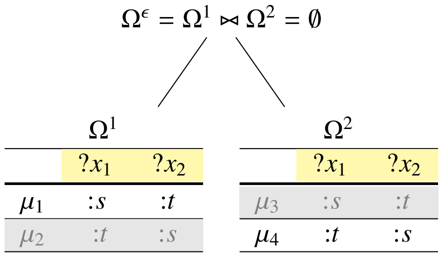
□

We shall call valuations that satisfy the antecedent of Lemma 9.2 *reducing* valuations.

Now, although reducing valuations preserves the information content, i.e. the semantics, of answer sets (and thus *a fortiori* of intermediate results in an evaluation tree *one-by-one*) it is not entirely obvious that equivalence will be propagated by joins up the tree. This is necessary if the reduction based on informativeness is to satisfy result set equivalence as required by Definition 6.2. Indeed, the just mentioned propagation property will have to be established, for as Example 9.3 shows, answer set-equivalence does not in the general case cater for joins:

Example 9.3. *Consider the evaluation tree shown in Fig. 21, where P is the union of the BGPs associated with the leaf nodes. The two solutions in Ω^1 are SPARQL-equivalent, modulo P, and likewise for the two solutions in Ω^2. Choosing arbitrarily between equivalent solutions from the same answer set, say, μ_1 in Ω^1 and μ_4 in Ω^2, can result in answers being lost, as indicated in the figure. That is, $\mu_1 \cup \mu_3$ and $\mu_2 \cup \mu_4$ are compatible,*

$$\begin{aligned}\Omega^1 &= [\![(?x_1, :p_1, ?x_2), (?x_2, :p_1, ?x_1)]\!]\\ \Omega^2 &= [\![(?x_1, :p_2, ?x_2), (?x_2, :p_2, ?x_1)]\!]\end{aligned}$$



	$?x_1$	$?x_2$
μ_1	$:s$	$:t$
μ_2	$:t$	$:s$

	$?x_1$	$?x_2$
μ_3	$:s$	$:t$
μ_4	$:t$	$:s$

Figure 21: Entailment is not a reduction operation

and hence would both be in Ω^ϵ for the unreduced tree. Yet μ_1 and μ_4 are clearly not compatible, since they bind different values to the live variables $\{?x, ?y\}$, and neither is μ_2 and μ_3. Thus, from the example it is clear that entailment on its own is not a reduction operation.

A significant consequence of this, in terms of federated evaluation, is that graph equivalence itself is not sufficient to cater for joins in a way that propagates equivalence to the final result.

Note that there are in general many distinct reducing valuations for a particular answer set. Given the present paper's concern with size, it is natural to seek the most effective ones among them, i.e. the valuations that yield the smallest reducts. The next set of results show that it is possible to kill two birds with one stone by selecting the reducing valuations that yield *lean* subsets, since as it turns out these valuations *do* cater for joins in the desired way.

The concept of leanness alluded to here is a straightforward lifting of the corresponding notion from [32].

Definition 9.7. Ω is *lean* if there is no valuation v s.t. $v(\Omega) \subset \Omega$. ∘

A lean set cannot be further reduced without somehow compromising information. We will define the lean answer sets wrt. an answer set Ω, denoted $\mathscr{L}(\Omega)$, as the family of lean subsets of Ω that preserves Ω by way of valuations. Formally:

Definition 9.8.

$$\mathscr{L}(\Omega) =_{def} \{\Omega' \subseteq \Omega\colon \Omega' \text{ is } \textit{lean}\text{, and } v(\Omega) = \Omega' \text{ for some valuation } v\}$$

∘

Corollary 9.3. *If Ω is an answer set, and P a query, then*

$$\mathscr{L}(\Omega) \subseteq [\Omega]_{\dashv\vdash_P}$$

We are now able to show that if answer sets do not share blank nodes, as is the case of two intermediate nodes in an evaluation tree that are not along the same path (ref. the *separation assumption* in Section 5) then lean answers caters for joins in the right way, propagating equivalence upwards.

Lemma 9.4. *Let $v^i(\Omega^i) \in \mathscr{L}(\Omega^i)$, $v^j(\Omega^j) \in \mathscr{L}(\Omega^j)$, and $ran(\Omega^i) \cap ran(\Omega^j) \cap B = \emptyset$, then*

$$\mathscr{L}(v^i(\Omega^i) \bowtie v^j(\Omega^j)) \subseteq \mathscr{L}(\Omega^i \bowtie \Omega^j)$$

Proof. We will show that there is a valuation v^+ s.t.

1. $v^i(\Omega^i) \bowtie v^j(\Omega^j) = v^+(\Omega^i \bowtie \Omega^j)$
2. $v^+(\Omega^i \bowtie \Omega^j) \subseteq \Omega^i \bowtie \Omega^j$
3. $\mathscr{L}(v^+(\Omega^i \bowtie \Omega^j)) \subseteq \mathscr{L}(\Omega^i \bowtie \Omega^j)$

whence substituting in (1) into (3) yields

$$\mathscr{L}(v^i(\Omega^i) \bowtie v^j(\Omega^j)) \subseteq \mathscr{L}(\Omega^i \bowtie \Omega^j)$$

Item (1) says that the combination of v^i and v^j is a valuation. Item (2) says that this valuation is a reducing valuation. The output of v^+ may not be lean. However, any lean subset of the output would also be a lean subset of the original join, which is item (3).

Now, we first note that since $ran(\Omega^i) \cap ran(\Omega^j) \cap B = \emptyset$, then $v^i_{|ran(\Omega^i)\cap B} \cup v^j_{|ran(\Omega^j)\cap B}$ is functional. Let v^+ be any valuation s.t. $v^+ \supseteq v^i_{|ran(\Omega^i)\cap B} \cup v^j_{|ran(\Omega^j)\cap B}$.

Part 1. Let $\mu_i \in \Omega^i$ and $\mu_j \in \Omega^j$ and $\mu_i \leftrightharpoons \mu_j$. Then

$$\begin{aligned} v^+(\mu_i \cup \mu_j) &= (v^+(\mu_i \cup \mu_j))_{|dom(\mu_i)} \cup (v^+(\mu_i \cup \mu_j))_{|dom(\mu_j)}\\ &= (v^i(\mu_i \cup \mu_j))_{|dom(\mu_i)} \cup (v^j(\mu_i \cup \mu_j))_{|dom(\mu_j)}\\ &= v^i(\mu_i) \cup v^j(\mu_j)\end{aligned}$$

Part 2. As a consequence of Definition 9.8, $v^i(\Omega^i) \subseteq \Omega^i$ and $v^j(\Omega^j) \subseteq \Omega^j$, hence $v^i(\Omega^i) \bowtie v^j(\Omega^j) \subseteq \Omega^i \bowtie \Omega^j$. From Part 1 we have that $v^i(\Omega^i) \bowtie v^j(\Omega^j) = v^+(\Omega^i \bowtie \Omega^j)$, thus by the substitution property of equality we have $v^+(\Omega^i \bowtie \Omega^j) \subseteq \Omega^i \bowtie \Omega^j$.

Part 3. Assume $\Omega^l \in \mathscr{L}(v^+(\Omega^i \bowtie \Omega^j))$. Then, as a consequence of Definition 9.8, there is a v s.t. $v(v^+(\Omega^i \bowtie \Omega^j)) = \Omega^l$ and $v(v^+(\Omega^i \bowtie \Omega^j)) \subseteq v^+(\Omega^i \bowtie \Omega^j)$. Then $v \circ v^+$ is a valuation s.t. $v \circ v^+(\Omega^i \bowtie \Omega^j) = \Omega^l$. Since Ω^l is lean, and $v \circ v^+(\Omega^i \bowtie \Omega^j) \subseteq v^+(\Omega^i \bowtie \Omega^j) \subseteq \Omega^i \bowtie \Omega^j$ as a consequence of Part 2, then by Definition 9.8 we have that $\Omega^l \in \mathscr{L}(\Omega^i \bowtie \Omega^j)$.

□

We define a straightforward reduction operation based on lean answers as follows:

Definition 9.9.

$$\delta^\Psi(\Omega) \in \mathscr{L}(\Omega)$$

∘

That is, we pick arbitrarily from $\mathscr{L}(\Omega)$. The following lemma verifies that reducing an evaluation tree with δ yields an answer set that is lean.

Lemma 9.5. *Let Ψ be an evaluation tree. Then for any $\Omega^i \in \Psi$ we have*

$$\Omega^i_\delta \in \mathscr{L}(\Omega^i)$$

Proof. Proof proceeds by induction on the depth of Ψ.

Base case: $d(\Psi) = 0$. Then Ψ contains only one node Ω^ϵ, and that node is a leaf. By the first case of Definition 6.1, and the definition of δ^Ψ, we have that $\Omega^\epsilon_\delta = \delta^\Psi(\Omega^\epsilon) \in \mathscr{L}(\Omega^\epsilon)$.

Induction step: $d(\Psi) > 0$. Suppose that $\Omega^i = \Omega^j \bowtie \Omega^k$. We need to show that $\left(\Omega^j \bowtie \Omega^k\right)_\delta \in \mathscr{L}(\Omega^j \bowtie \Omega^k)$. By the second case of Definition 6.1, we have that $\left(\Omega^j \bowtie \Omega^k\right)_\delta = \delta^\Psi_i\left(\Omega^j_\delta \bowtie \Omega^k_\delta\right)$, hence it suffices to show that

$$\delta^\Psi_i\left(\Omega^j_\delta \bowtie \Omega^k_\delta\right) \in \mathscr{L}(\Omega^j \bowtie \Omega^k) \tag{3}$$

Assume for induction hypothesis that $\Omega^j_\delta \in \mathscr{L}(\Omega^j)$ and $\Omega^k_\delta \in \mathscr{L}(\Omega^k)$. By Lemma 9.4, we have that $\mathscr{L}(\Omega^j_\delta \bowtie \Omega^k_\delta) \subseteq \mathscr{L}(\Omega^j \bowtie \Omega^k)$, since $i \nleq j$ and $i \ngeq j$ thus $ran(\Omega^j) \cap ran(\Omega^k) \cap B = \emptyset$. Finally, since $\delta^\Psi_i\left(\Omega^j_\delta \bowtie \Omega^k_\delta\right) \in \mathscr{L}(\Omega^j_\delta \bowtie \Omega^k_\delta)$ by the definition of δ (Definition 9.9), it follows that $\delta^\Psi_i\left(\Omega^j_\delta \bowtie \Omega^k_\delta\right) \in \mathscr{L}(\Omega^j \bowtie \Omega^k)$.

□

For proving that δ is a reduction operation, however, we must establish that answer set equivalence (wrt. SPARQL equivalence) holds after the final extraction of projected variables—i.e. we must prove result set equivalence. The following lemma establishes something slightly stronger than required.; namely that a) informativeness is preserved, and b) the reduced answer is a subset of the unreduced.

Lemma 9.6. *For evaluation tree Ψ and any $W \in dom(\Omega^\epsilon)$, we have that*

1. *$\pi_W(\Omega^\epsilon_\delta)$ preserves information in $\pi_W(\Omega^\epsilon)$, and*
2. *$\pi_W(\Omega^\epsilon_\delta) \subseteq \pi_W(\Omega^\epsilon)$*

Proof. By Lemma 9.5, $\Omega^\epsilon_\delta \in \mathscr{L}(\Omega^\epsilon)$, hence there is a valuation v s.t. $v(\Omega^\epsilon) = \Omega^\epsilon_\delta$, and $\Omega^\epsilon_\delta \subseteq \Omega^\epsilon$.

Part 1. We will show that v is a valuation s.t. $v(\pi_W(\Omega^\epsilon)) = \pi_W(\Omega^\epsilon_\delta)$. From the definition of π_W, and the assumption that $v(\Omega^\epsilon) = \Omega^\epsilon_\delta$, it suffices to show that $(v(\Omega^\epsilon))_{|W} = v(\Omega^\epsilon_{|W})$.

($\Rightarrow$) Assume $\mu' \in (v(\Omega^\epsilon))_{|W}$, hence there is a $\mu \in v(\Omega^\epsilon)$ s.t. $\mu \supseteq \mu'$ and a $\mu^* \in \Omega^\epsilon$ s.t. $v(\mu^*) = \mu$. Immediately, we have that $\mu^*_{|W} \in \Omega^\epsilon_{|W}$. Now, since $\mu^*_{|W} \subseteq \mu^*$ and v is functional it follows from $v(\mu^*) = \mu \in v(\Omega^\epsilon)$ that $v(\mu^*_{|W}) \in v(\Omega^\epsilon_{|W})$. Hence it only remains to show that $v(\mu^*_{|W}) = \mu'$. But this follows from $dom(v(\mu^*_{|W})) = dom(\mu^*_{|W}) = dom(\mu_{|W})$ and the functionality of v.

($\Leftarrow$) Assume $\mu' \in v(\Omega^\epsilon_{|W})$. Hence there is a $\mu^*_{|W} \in \Omega^\epsilon_{|W}$ s.t. $v(\mu^*_{|W}) = \mu'$ and $\mu^* \in \Omega^\epsilon$. Then $v(\mu^*) \in v(\Omega^\epsilon)$, and since $v(\mu^*_{|W}) \subseteq v(\mu^*)$, it follows that $v(\mu^*_{|W}) = (v(\mu^*))_{|W}$ by equality of domains. Now, since $v(\mu^*_{|W}) = \mu'$ and $v(\mu^*_{|W}) = (v(\mu^*))_{|W}$, we have that $(v(\mu^*))_{|W} = \mu'$, and since $v(\mu^*) \in v(\Omega^\epsilon)$, it follows that $\mu' \in (v(\Omega^\epsilon))_{|W}$.

Part 2. Since $\Omega^\epsilon_\delta \subseteq \Omega^\epsilon$, then it follows directly that $(\Omega^\epsilon_\delta)_{|W} \subseteq (\Omega^\epsilon)_{|W}$ hence $\pi_W(\Omega^\epsilon_\delta) \subseteq \pi_W(\Omega^\epsilon)$.

□

Now, that δ satisfies the shrinking property of Definition 6.2 follows directly from Definition 9.8. Hence in order to establish that δ is a reduction operator, we must show that it yields result equivalence.

Corollary 9.7. *Let Ψ be a tree and P a query, then*

$$\pi_W(\Omega^\epsilon_\delta) \in [\pi_W(\Omega^\epsilon)]_{\dashv\vdash_P}$$

Proof. Directly from Lemmas 9.2 and 9.6. □

Hence δ is a valid reduction operation under SPARQL equivalence. As a side remark, we note that the resulting answer set is not itself necessarily lean, as the project operation can produce new redundancies.

9.3. A characterization in terms of database completions

We shall return briefly to the topic of the relation between the concept of result-set equivalence and the concept of database completions [4, 5]. Recall that the former is a graph-theoretic concept that relates two answer set by way of the respective induced graphs. The second concept, on the other hand, is the concept of a function providing one of several possible meanings of an incomplete database by "plugging its holes" i.e. substituting concrete values for nulls. Although quite different on the face of it, there is a gratifyingly simple relationship between the two expressed in Theorem 9.8.

Theorem 9.8.

$$[\![\pi_W(\Omega^\epsilon_\delta)]\!]_{OWA} = [\![\pi_W(\Omega^\epsilon)]\!]_{OWA}$$

Proof.

($\Rightarrow$) If $\Omega' \in [\![\pi_W(\Omega^\epsilon_\delta)]\!]_{OWA}$ then by the definition of *open-world semantics* there is a valuation $v^r(\pi_W(\Omega^\epsilon_\delta)) \subseteq \Omega'$ where both $v^r(\pi_W(\Omega^\epsilon_\delta))$ and Ω' are free from blank nodes. By Lemma 9.6, there is a valuation $v(\pi_W(\Omega^\epsilon)) = \pi_W(\Omega^\epsilon_\delta)$, hence $v^r \circ v$ is a valuation s.t. $v^r \circ v(\pi_W(\Omega^\epsilon)) \subseteq \Omega'$ and $v^r \circ v(\pi_W(\Omega^\epsilon))$ is free from blank nodes, thus $\Omega' \in [\![\pi_W(\Omega^\epsilon)]\!]_{OWA}$.

($\Leftarrow$) If $\Omega' \in [\![\pi_W(\Omega^\epsilon)]\!]_{OWA}$ then there is a valuation $v(\pi_W(\Omega^\epsilon)) \subseteq \Omega'$ where both $v(\pi_W(\Omega^\epsilon))$ and Ω' are free of blank nodes. Since $\pi_W(\Omega^\epsilon_\delta) \subseteq \pi_W(\Omega^\epsilon)$ then it follows that $v(\pi_W(\Omega^\epsilon_\delta)) \subseteq v(\pi_W(\Omega^\epsilon))$ thus also $v(\pi_W(\Omega^\epsilon_\delta))$ is free of blank nodes and $v(\pi_W(\Omega^\epsilon_\delta)) \subseteq \Omega'$ hence $\Omega' \in [\![\pi_W(\Omega^\epsilon_\delta)]\!]_{OWA}$.

□

Explained in words, Theorem 9.8 says that the root of an evaluation tree Ψ and the root of its δ-reduction Ψ_δ have the same database completions under the open world semantics. The theorem, one might say, sanctions the concept of SPARQL-equivalence by showing that two equivalent result sets also have the same completions in a relational sense. As such, it may be taken as independent validation of the present approach.

It is interesting that the presupposition of open world semantics is necessary for this result to hold, for as Example 9.4 demonstrates it breaks under *closed world* semantics.

Example 9.4. *Consider answer sets $\pi_W(\Omega^\epsilon)$ and Ω' s.t.*

$$\pi_W(\Omega^\epsilon) = \begin{array}{cc} \hline ?x & ?y \\ \hline :s & \_:b_1 \\ \hline :s & \_:b_2 \\ \hline \end{array} \qquad \Omega' = \begin{array}{cc} \hline ?x & ?y \\ \hline :s & :t \\ \hline :s & :u \\ \hline \end{array}$$

and let δ choose such that

$$\pi_W(\Omega^\epsilon_\delta) = \begin{array}{cc} \hline ?x & ?y \\ \hline :s & \_:b_1 \\ \hline \end{array}$$

Then $\Omega' \in [\![\pi_W(\Omega^\epsilon)]\!]_{CWA}$, but $\Omega' \notin [\![\pi_W(\Omega^\epsilon_\delta)]\!]_{CWA}$ since there exists no valuation v that can make $v(\pi_W(\Omega^\epsilon_\delta)) = \Omega'$. That is, any such v would have to map $\_:b_1$ to both :t and :u, hence not a function.

As mentioned at the start of the present section, the essential open world nature of Theorem 9.8 reflects a duality between respectively reductions, which by definition *remove*, and completions, which under the open world semantics are allowed to *add* tuples. Stated differently, since a δ-reduct $\Omega^\epsilon_\delta \in \Psi_\delta$ is SPARQL-equivalent to $\Omega^\epsilon \in \Psi$ the former, by the theorem 9.8, generates the same set of completions as the latter under the open world semantics. This in turn just goes to show that the reduced result-set is a smaller version of the original set not only in terms of SPARQL equivalence but also in terms of informativeness as that concept is defined in relational database theory.

10. Compositions and size

In Section 6 we remarked that a reduced tree Ψ_o for some operation o and an evaluation tree Ψ, may not itself be an evaluation tree. Take the operation τ: Ψ_τ need not be an evaluation tree because it is not a distributive operation. This was shown in Example 7.2. This means that an interior node in Ψ_τ is not in the general case equal to the join of its left and right subtrees, but rather to the *truncation* of that join.

One consequence of this is that compositions of reduction operators, do not type check. As the definitions stand, one cannot pipe the output of one reduction into the input of another because all operators expect *evaluation trees* as inputs. To make reduction operators composable, the key definitions need to be pushed one step up the ladder of abstraction.

For that purpose, we shall in this section talk in abstract terms about *binary operator trees*, or equivalently about binary $\otimes$-trees where $\otimes$ may but need not be $\bowtie$. The point of this is that although reduction operators may not output *evaluation* trees, they do output binary operator trees in the more abstract sense. The operation σ outputs an evaluation tree, τ outputs a binary operator tree where $\otimes = \tau \bowtie$—that is, a tree under the binary operator formed from the composition of join and truncation—and δ outputs a binary operator tree where $\otimes = \delta\bowtie$.

In general, we make no assumptions about $\otimes$ except that it satisfies the shrinking property from Section 6.2. The concept of the o-reduction of a binary $\otimes$-tree is an accordingly generalized version of Definition 6.1:

Definition 10.1 (o-reduced $\otimes$-tree)**.** Let o be an operation of type $o : \mathbb{T} \times A^* \times \mathscr{A} \longrightarrow \mathscr{A}$. The *reduct* Ψ_o of a binary $\otimes$-tree Ψ, is a binary $o\otimes$-tree derived from Ψ in the following manner:

$$\Omega^i_o =_{df} \begin{cases} o^\Psi_i(\Omega^i) & \text{if } \Omega^i \text{ is a leaf} \\ \\ o^\Psi_i(\Omega^j_o \otimes \Omega^k_o) & \text{if } \Omega^i = \Omega^j \otimes \Omega^k \end{cases}$$

∘

In complete analogy to Definition 6.2, we shall then say that o is a *composable reduction operator* iff it satisfies result equivalence wrt. binary operator trees in general. We have

Theorem 10.1. *The operators σ, δ and τ are all composable reduction operations wrt. SPARQL equivalence, and so is any composition of them.*

Sketch of proof. Composable reduction operators have the same input and return types by design. It follows that if each of σ, δ and τ can be shown to be a composable reduction operation, then all compositions of them are (composable) reduction operators too. Hence it suffices to show the former. The proof is a straightforward, but rather tedious rerun, of the corresponding proofs for the special case of $\bowtie$-trees that we have already given. As an illustration we show how to lift Lemma 7.1 and Lemma 7.2 to binary operation trees.

For stability (Lemma 7.1), let Ψ be a $\otimes$-tree and $\Omega^i = \Omega^j \otimes \Omega^k \in \Psi$. We need to show that

$$\pi_{V_i}(\pi_{V_j}(\Omega^j) \otimes \pi_{V_k}(\Omega^k)) = \pi_{V_i}(\Omega^j \otimes \Omega^k)$$

Let $\mu \in \pi_{V_i}(\pi_{V_j}(\Omega^j) \otimes \pi_{V_k}(\Omega^k))$. Since $\otimes$ satisfies shrinking we have $\pi_{V_i}(\pi_{V_j}(\Omega^j) \otimes \pi_{V_k}(\Omega^k)) \subseteq \pi_{V_i}(\pi_{V_j}(\Omega^j) \bowtie \pi_{V_k}(\Omega^k))$ whence $\mu = \pi_{V_i}(\pi_{V_j}(\mu_j) \cup \pi_{V_k}(\mu_k))$ for some μ_j and μ_k. The rest of the verification is exactly like that for Lemma 7.1.

For Lemma 7.2 it suffices to check that the induction step goes through: suppose that $\Omega^i = \Omega^j \otimes \Omega^k$, and assume as induction hypothesis that $\Omega^j_\tau = \pi_{V_j}(\Omega^j)$ and $\Omega^k_\tau = \pi_{V_k}(\Omega^k)$. We need to show that $\Omega^i_\tau = \pi_{V_i}(\Omega^j \otimes \Omega^k)$. We have

$$\begin{aligned} \Omega^i_\tau &= \tau^\Psi_i(\Omega^j_\tau \otimes \Omega^k_\tau) && \text{df. } \Omega_\tau \\ &= \tau^\Psi_i(\pi_{V_j}(\Omega^j) \otimes \pi_{V_k}(\Omega^k)) && \text{by ind. hyp.} \\ &= \pi_{V_i}(\pi_{V_j}(\Omega^j) \otimes \pi_{V_k}(\Omega^k)) && \text{by def. of } \tau \\ &= \pi_{V_i}(\Omega^j \otimes \Omega^k) && \text{by stability} \end{aligned}$$

□

The importance of Theorem 10.1 is that it ensures that we can compose and reduction operators freely and apply them repeatedly without worrying that the overall semantics of the federated evaluation process will be skewed.

Nevertheless, different compositions do not in general yield *identical* results, nor do they yield equally small ones. Consider the graph in Fig. 22 . Paths in the graph represent different compositions of reduction operators, and the nodes are the results. J stands for live join variables, Π for project variables, and I for idle variables. We denote the answer set whose only incoming edge is a loop by Ω for easy reference. We think of this as an unreduced answer set in some evaluation tree. We make some observation regarding the order in which reduction operators are applied.

Note first that all of the end results—i.e. all nodes with outdegree 0—are SPARQL equivalent modulo the graph pattern P. Nevertheless, they are all different, and only two of them have the same size. Next, going into specifics we note that the δ operation tends to have a lower reduction rate when it is applied early rather than late. This is because the concept of informativeness is a *semantic* one, more precisely because the concept of a valuation is defined in such a way that it respects semantic relationships between rows in the answer set. E.g. in the set Ω all rows are interrelated by blank nodes in such a way that no row can be reduced without altering the semantics of Ω. If any row is removed then a cross-reference between blank nodes disappears and the connection it expresses is no longer implicit in the answer set. In contrast we may think of σ and τ as *syntactic* operators that remove rows and columns respectively based purely on the form of the given tree (τ) and the position of blank nodes in intermediate results (σ). Not being sensitive to semantics, these operators will sometimes sever the links between rows by removing an occurrence of a blank node, thus making the resulting set less constrained. It makes sense therefore to expect smaller intermediate results if the syntactic operators are applied before δ since delta preserves such constraints. Fig. 22 confirms this, we have

$$\delta\sigma\tau(\Omega) < \tau\delta\sigma(\Omega) \leq \tau\delta\sigma(\Omega) < \tau\sigma\delta(\Omega)$$

The composition that produces the least intermediate result is that which applies both of the syntactic operators first, and δ last. We conjecture that this holds in general:

Theorem 10.2 (Row minimality)**.** *Let θ be any reduction operator and Ψ an evaluation tree. Then if for any node $\Omega^k \in \Psi$ it holds that $dom(\Omega^k_\theta) \subset dom(\Omega^k_{\delta\sigma\tau})$ for some $k \in dom(\Psi)$, then θ does not satisfy SPARQL result equivalence.*

Also worth noting is the fact that the operation $\sigma\delta\tau$ removes strictly more rows than the combination $\sigma\delta$, despite the fact that τ is not a row operator. This can be verified by following the corresponding paths in Fig. 22. This tells us that there is a genuine synergy between row and column operators. The explanation is not too difficult to spot: when truncation removes idle columns, it may happen to remove RDF terms and thereby to generalize rows. As a consequence, some rows may collapse under the relation of informativeness leaving a kernel with fewer rows.

11. Conclusion

We have defined an abstract notion of a reduction operation by stipulating that it should a) behave in a way that allows reductions to be recursively applied to the nodes in an evaluation tree, b) avoid increasing the size of an answer set, and c) yield final results that are equivalent to that produced by the original query wrt. some suitable equivalence relation. The point of the abstract definition is to allow vertical and horizontal operations to be studied in combination.

The equivalence relation must of course be meaningful to be of interest. For the general case we have proposed one based on entailment between query patterns that we have called *SPARQL*-equivalence. This concept is in turn based on a concept of graph homomorphisms that generalizes *RDF*-homomorphisms by permitting uninstantiated variables in a graph pattern.

We have further formalized row- and column-reducing operations and proved that they conform to the abstract definition. These operations are all intended for *zero-knowledge* federation, meaning that they express heuristic rules that can be applied without looking at anything else than the shape of an evaluation tree and the distribution of variables within it. In particular, these reduction operators are not sensitive to the distribution of data in a given tree.

Finally, we have studied the composition of these operators and determined that different orderings gives different effect. It seems that in general syntactic operators based on the position of a variable in a tree must be applied before semantic ones that factor in logical content such as cross-references between rows. The issue is left open as at the time of writing we have no general minimality theorem to offer.

Due to their zero-knowledge nature, the reduction operations developed in this paper share the commonality that redundancies can be computed without scanning solutions upwards or sideways in the evaluation tree, and without any other form of coordination between sources. These operators ought to be useful for query federation in dynamic network topologies, and useful for more traditional distributed, share-nothing architectures as a means of reducing coordination, communication and memory consumption.

$$P = \{(?x, :p, ?y), (?x, :p, ?z)\},\ \ \Omega = [\![P]\!]$$

$$\begin{tikzcd}
\Omega = \begin{array}{ccc} \mathrm{J} & \Pi & \mathrm{I} \\ \hline ?\mathrm{x} & ?\mathrm{y} & ?\mathrm{z} \\ \hline\hline :u & :s & \_:b_1 \\ :u & \_:b_2 & \_:b_3 \\ \_:b_2 & :s & :t \\ :u & \_:b_1 & :t \\ \hline \end{array} \arrow[loop above, "\delta"] \arrow[r, "\sigma"] \arrow[d, "\tau"'] \arrow[ddd, bend right, "\sigma"'] & \begin{array}{ccc} \mathrm{J} & \Pi & \mathrm{I} \\ \hline ?\mathrm{x} & ?\mathrm{y} & ?\mathrm{z} \\ \hline\hline :u & :s & \_:b_1 \\ :u & \_:b_2 & \_:b_3 \\ :u & \_:b_1 & :t \\ \hline \end{array} \arrow[r, "\tau"] & \begin{array}{cc} \mathrm{J} & \Pi \\ \hline ?\mathrm{x} & ?\mathrm{y} \\ \hline\hline :u & :s \\ :u & \_:b_2 \\ :u & \_:b_1 \\ \hline \end{array} \\
\begin{array}{cc} \mathrm{J} & \Pi \\ \hline ?\mathrm{x} & ?\mathrm{y} \\ \hline\hline :u & :s \\ :u & \_:b_2 \\ \_:b_2 & \_:s \\ :u & \_:b_1 \\ \hline \end{array} \arrow[r, "\delta"] \arrow[dr, "\sigma"] & \begin{array}{cc} \mathrm{J} & \Pi \\ \hline ?\mathrm{x} & ?\mathrm{y} \\ \hline\hline :u & :s \\ :u & \_:b_2 \\ \_:b_2 & \_:s \\ \hline \end{array} \arrow[r, "\sigma"] & \begin{array}{cc} \mathrm{J} & \Pi \\ \hline ?\mathrm{x} & ?\mathrm{y} \\ \hline\hline :u & :s \\ :u & \_:b_2 \\ \hline \end{array} \\
 & \begin{array}{cc} \mathrm{J} & \Pi \\ \hline ?\mathrm{x} & ?\mathrm{y} \\ \hline\hline :u & :s \\ :u & \_:b_2 \\ :u & \_:b_1 \\ \hline \end{array} \arrow[r, "\delta"] & \begin{array}{cc} \mathrm{J} & \Pi \\ \hline ?\mathrm{x} & ?\mathrm{y} \\ \hline\hline :u & :s \\ \hline \end{array} \\
\begin{array}{ccc} \mathrm{J} & \Pi & \mathrm{I} \\ \hline ?\mathrm{x} & ?\mathrm{y} & ?\mathrm{z} \\ \hline\hline :u & :s & \_:b_1 \\ :u & \_:b_2 & \_:b_3 \\ :u & \_:b_1 & :t \\ \hline \end{array} \arrow[r, "\delta"] & \begin{array}{ccc} \mathrm{J} & \Pi & \mathrm{I} \\ \hline ?\mathrm{x} & ?\mathrm{y} & ?\mathrm{z} \\ \hline\hline :u & :s & \_:b_1 \\ :u & \_:b_1 & :t \\ \hline \end{array} \arrow[r, "\tau"] & \begin{array}{cc} \mathrm{J} & \Pi \\ \hline ?\mathrm{x} & ?\mathrm{y} \\ \hline\hline :u & :s \\ :u & \_:b_1 \\ \hline \end{array}
\end{tikzcd}$$

Figure 22: Compositions of δ, σ and τ

References

[1] A. Potter, B. Motik, Y. Nenov, I. Horrocks, Distributed RDF Query Answering with Dynamic Data Exchange, in: International Semantic Web Conference, Springer, 480–497, 2016.

[2] P. J. Hayes, P. F. Patel-Schneider, RDF 1.1 Semantics. W3C Recommendation, February 2014, World Wide Web Consortium. Retrieved from https://www. w3. org/TR/2014/REC-rdf11-mt-20140225 .

[3] A. Stolpe, J. Halvorsen, Distributed query processing in the presence of blank nodes, Semantic Web 8 (6) (2017) 1001–1021.

[4] L. Libkin, Incomplete data: what went wrong, and how to fix it, ACM, 1–13, 2014.

[5] L. Libkin, Certain answers as objects and knowledge, Artificial Intelligence 232 (2016) 1 – 19.

[6] M. Schmidt, M. Meier, G. Lausen, Foundations of SPARQL query optimization, in: Database Theory - ICDT 2010, 13th International Conference, Lausanne, Switzerland, March 23-25, 2010, Proceedings, 4–33, 2010.

[7] C. Buil-Aranda, M. Arenas, O. Corcho, A. Polleres, Federating queries in SPARQL 1.1: Syntax, semantics and evaluation, Web Semantics: Science, Services and Agents on the World Wide Web 18 (1) (2013) 1–17.

[8] C. Buil-Aranda, A. Polleres, J. Umbrich, Strategies for executing federated queries in SPARQL1. 1, in: International Semantic Web Conference, Springer, 390–405, 2014.

[9] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, D. Reynolds, SPARQL basic graph pattern optimization using selectivity estimation, in: Proceedings of the 17th international conference on World Wide Web, ACM, 595–604, 2008.

[10] P. Tsialiamanis, L. Sidirourgos, I. Fundulaki, V. Christophides, P. Boncz, Heuristics-based query optimisation for SPARQL, in: Proceedings of the 15th International Conference on Extending Database Technology, ACM, 324–335, 2012.

[11] O. Görlitz, S. Staab, SPLENDID: SPARQL Endpoint Federation Exploiting VoID Descriptions, in: Proceedings of the Second International Conference on Consuming Linked Data-Volume 782, CEUR-WS. org, 13–24, 2011.

[12] A. Gubichev, T. Neumann, Exploiting the query structure for efficient join ordering in SPARQL queries., in: EDBT, vol. 14, 439–450, 2014.

[13] A. Schwarte, P. Haase, K. Hose, R. Schenkel, M. Schmidt, FedX: Optimization Techniques for Federated Query Processing on Linked Data, in: The Semantic Web–ISWC 2011, Springer, 601–616, 2011.

[14] P. Peng, L. Zou, M. T. Özsu, L. Chen, D. Zhao, Processing SPARQL queries over distributed RDF graphs, The VLDB Journal 25 (2) (2016) 243–268.

[15] M. Saleem, A.-C. N. Ngomo, J. X. Parreira, H. F. Deus, M. Hauswirth, DAW: Duplicate-AWare Federated Query Processing over the Web of Data, in: International Semantic Web Conference, Springer, 574–590, 2013.

[16] G. Montoya, H. Skaf-Molli, P. Molli, M.-E. Vidal, Federated SPARQL queries processing with replicated fragments, in: International Semantic Web Conference, Springer, 36–51, 2015.

[17] M. Saleem, A.-C. N. Ngomo, HiBISCuS: Hypergraph-Based Source Selection for SPARQL Endpoint Federation, in: European Semantic Web Conference, Springer, 176–191, 2014.

[18] M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, E. Ruckhaus, ANAPSID: an adaptive query processing engine for SPARQL endpoints, The Semantic Web–ISWC 2011 (2011) 18–34.

[19] M. Arenas, C. Gutierrez, J. Pérez, Foundations of RDF Databases, in: S. Tessaris, E. Franconi, T. Eiter, C. Gutierrez, S. Handschuh, M.-C. Rousset, R. A. Schmidt (Eds.), Reasoning Web. Semantic Technologies for Information Systems, vol. 5689 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 158–204, 2009.

[20] S. Gorn, Explicit definitions and lingustic dominoes, in: J. Hart, S. Takasu (Eds.), Systems and computer Science, University of Toronto Press, 77–105, 1967.

[21] A. Stolpe, A logical characterisation of SPARQL federation, Semantic Web Journal 6 (6) (2014) 565–584.

[22] A. van Aggelen, L. Hollink, M. Kemman, M. Kleppe, H. Beunders, The debates of the European parliament as linked open data, Semantic Web 8 (2) (2017) 271–281.

[23] B. Quilitz, U. Leser, Querying distributed RDF data sources with SPARQL, in: Proc. ESWC' 08, 2008.

[24] P. A. Bernstein, D.-M. W. Chiu, Using Semi-Joins to Solve Relational Queries, J. ACM 28 (1) (1981) 25–40.

[25] P. A. Bernstein, N. Goodman, The Power of Natural Semijoins 10 (4) (1981) 751–771.

[26] A. Silberschatz, H. Korth, S. Sudarshan, Database Systems Concepts, McGraw-Hill, Inc., New York, NY, USA, 5 edn., ISBN 0072958863, 9780072958867, 2006.

[27] P. G. Kolaitis, M. Y. Vardi, Conjunctive-Query Containment and Constraint Satisfaction, Journal of Computer and System Sciences 61 (2) (2000) 302 – 332.

[28] G. Gottlob, N. Leone, F. Scarcello, Hypertree Decompositions and Tractable Queries, Journal of Computer and System Sciences 64 (3) (2002) 579 – 627.

[29] V. Dalmau, P. G. Kolaitis, M. Y. Vardi, Constraint Satisfaction, Bounded Treewidth, and Finite-Variable Logics, in: CP, 310–326, 2002.

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

[31] J.-F. Baget, RDF Entailment as a Graph Homomorphism, Springer Berlin Heidelberg, 82–96, 2005.

[32] C. Gutierrez, C. A. Hurtado, A. O. Mendelzon, J. Pérez, Foundations of Semantic Web Databases, Journal of Computer and System Sciences 77 (3) (2011) 520–541.