Undefined 1 (2009) 1–5
IOS Press

Distributed query processing in the presence of blank nodes.

Audun Stolpe, Jonas Halvorsen,
Norwegian Defence Research Establishment (FFI), Postboks 25, 2027 Kjeller, Norway
E-mail: audun.stolpe@ffi.no, jonas.halvorsen@ffi.no

Abstract. This paper demonstrates that the presence of blank nodes in RDF data represents a problem for distributed processing of SPARQL queries. It is shown that the usual decomposition strategies from the literature will leak information—even when information derives from a single source. It is argued that this leakage, and the proper reparational measures, need to be accounted for in a formal semantics. To this end, the standard set-based SPARQL 1.1. semantics is generalized with a parameter representing execution contexts. This makes it possible to keep tabs on the naming of blank nodes across execution contexts, which in turn makes it possible to articulate a decomposition strategy that is provably sound and complete wrt. any selection of RDF sources even when blank nodes are allowed. Alas, this strategy is not computationally tractable. However, there are ways of utilizing knowledge about the sources, if one has it, that will help considerably.

Keywords: SPARQL federation, distributed query processing, logical foundation, completeness, decomposition scheme

1. Introduction

This paper is concerned with giving a general theoretical foundation for the semantics of distributed SPARQL processing. It was inspired in part by the observation that whilst a number of distributed SPARQL processors exist already, foundational work that allows one to study their relative behaviour in a principled manner is still in short supply.

In most ways, the present study can be regarded as a sequel to [39], in which the concept of a federation scheme was first introduced. Essentially, a federation scheme is a theoretical device that captures formally the conditions under which a particular federation strategy will yield sound and complete answer sets. Contrapositively, it can be regarded as a way of characterizing the behaviour of the federation strategy in question by making theoretically explicit when it will miss answers and why. While [39] gave several examples of sound and complete ways of decomposing and evaluating a query over a set of sources, these results were valid under the proviso that the contributing sources do not employ blank nodes for encoding information. The main contribution of the present paper is to remove this restriction thus giving, for the first time we believe, a fully general declarative semantics for distributed processing of basic graph patterns in SPARQL.

There is a singular tension in the Resource Description Framework between, on the one hand, the strictly local validity of blank nodes and, on the other, the global pretensions of the data model. RDF affords a knowledge representation language for making assertions that are meant to be valid in a global or Web-wide scope. Yet, it recognizes the existence of entities that cannot be referenced outside their source of origin, and that cannot be identified across query execution contexts.

The Semantic Web community is deeply ambiguous in its attitude towards blank nodes. The Linked Data community generally shuns them—in Bizer they are discussed under the heading "RDF features best avoided in the Linked Data context" [19]. On the other hand, asserting the existence of something that one is not willing to name is generally recognized as indispensable for design patterns that rely on express-

ing general logical concepts such as e.g. n-ary relations, lists, sets or classes. However one sees it, there is no getting around the well-documented prevalence of blank nodes on the Web—a fact which is not likely to change anytime soon.

In rough outline, the problem with blank nodes is that if a federator is to combine information across SPARQL endpoints, then it will have to join suitably small fragments of information from different sources. This in turn involves evaluating suitably small subqueries to get at all the partial answers that are required for answering the global query in full. Alas, in the process patterns may happen to be broken up that match a partial answer in a single source where a blank node figures in join position. If so, then that partial answer cannot be put back together afterwards, because the blank nodes that were previously joined will no longer be identifiable as the same node. The situation poses somewhat of a dilemma, and resolving it requires rather profound, though not too complex, amendments to the SPARQL 1.1. semantics.

The layout and specific contributions of this paper are as follows: Section 3 identifies and describes the adverse effect of blank nodes on distributed query processing. Section 4 provides the semantic prerequisites for working out a solution to the said dilemma. This consists mainly in generalizing the SPARQL 1.1. semantics by adding a formal parameter for execution contexts to the notion of an answer set. The concept of an execution context is left largely implicit in the SPARQL 1.1. specification. In the distributed setting, it is a *condition sine qua non* for an adequate semantics and therefore needs to be formalized. Section 5 flexes the new semantical machinery by providing an existence theorem showing that if an answer to a query exists in a set of contributing sources taken as a whole, then there is a way to retrieve it even allowing for blank nodes. Section 6 reviews the federation-schemes framework as developed in [39] as a prerequisite for the next section. It does not contribute any new results per se, but merely adapts the relevant concepts and definitions to the generalized semantics developed in Section 4. Finally, Section 7 ties the threads together by abstracting from the existence theorem in Section 5 a federation scheme that gives sound and complete answer sets in the absence of any knowledge about the structure of the sources, blank nodes allowed. However this section also shows that there is a rather steep price to pay for ignorance when blank nodes are involved. More specifically, it is shown that the distributed query answering in the presence of blank nodes when no knowledge is available is inherently hard. Different ways to mitigate the situation are sketched based on obtaining knowledge either by *regimentation* of the sources *or* by probing them, pointing to interesting lines of future research.

2. Related work

A tripartite classification of approaches to SPARQL federation is explicit in Görlitz and Staab [14] and implicit in Betz et al. [8] and Hose et al. [21].

Lookup-based federation: Also known as federation-by-traversal, or follow-your-nose traversal [4], iteratively downloads and evaluates data based on links (usually as prescribed by the Linked Data principles [9]) from one dataset into another. The answer to a query is composed by cumulatively adding answers from incoming data during the traversal process. This approach is exemplified by e.g. [15,16,17,18] and [24].

Warehousing: Broadly understood, warehousing covers approaches that seek to collect or assemble all data of relevance ahead of query time into a respository that behaves as if it is a centralized single store. This area naturally shades off into cloud computing [36], distributed file systems, and big data technologies. Recent efforts in this direction focus on the use of cluster technology such as MapReduce and Hadoop e.g. [12,20,22,23,25] and [31], Spark e.g. [11] and GraphX e.g [35].

Distributed query processing: Distributed query processing relies on analysing a query to identify a set of relevant RDF graphs—with or without the aid of statistics—to which subqueries can be assigned. Like federation-by-traversal and unlike warehousing, queries are evaluated remotely. Examples of this approach include [3,7,29,33] and [38].

If the issue were pressed, the present paper would be most naturally classified as belonging to the field of distributed query processing. However, it does not really advocate ‘an approach’ to SPARQL federation. Rather, it aims to articulate and develop a set of semantical primitives in terms of which any federation engine ought to be describable. To the authors’ knowledge it is the first foundational study of distributed query processing that allots to blank nodes a clearly described role in a strictly logico-mathematical theory.

It should be said that the tripartite classification mentioned above is very rough, and cuts across important and interesting distinctions in the literature. Conversely, there are subfields within federation that cut across *it*. It can plausibly be argued for instance that *scalability* and *optimization* are issues that are largely independent, or at least not tightly coupled with, any one particular approach. On the contrary, a lot of work seems to be going on here with more general significance.

To mention but a few, there is the area of *adaptive* query answering that explores how the evaluation of a query may be adjusted at run-time e.g. by reordering joins [2] or in order by exploiting approximate membership functions reducing HTTP requests [40]. There is also a fair bit of activity in the field of *caching*, *data replication* and *load balancing*. Thus, [28] explores the use of replicated fragments of data graphs in order to mimimize the size of intermediate results, whilst [41,42] adresses client-server load balancing through the use of so called *Triple Pattern Fragments (TPFs)*.

Finally, current trends in OBDA (*Ontology-based Data Access*) seem to be shifting emphasis onto OBDI (*Ontology-based Data Integration*). For instance [10] is concerned with the problem of how to utilize `owl:sameAs` links to extend query-rewriting to multiple sources. The interest in `owl:sameAs` statements overlaps a theme in [30], only whereas the latter can be seen as a wareousing approach the former is a distributed query processing approach.

3. Problem description.

It seems reasonable, by default at least, to require of a distributed query processor that it neither misses nor invents answers; all and only the answers that are warranted by the sum total of information contained in the contributing sources should, if there are no overriding reasons to the contrary, be returned.

To illustrate what is meant here by 'the sum total of information', consider the two RDF graphs in Figures 1 and 2 respectively, call them source A and B. As should be apparent, they encode information about trophies won by Roger Federer and Raphael Nadal in Grand Slam tournaments.[1] Now, according to source A the Wimbledon tournament is a Grand Slam tournament but the French Open is not. One may assume that this is an unintentional omission on behalf of the curator of the data set. The missing piece of information is provided by source B, however, so when the two sources are merged, as in Figure 3, all tennis events become appropriately classified. Therefore, the query in figure 4—which asks for the name of a player and the year in which he won a Grand Slam tournament—when evaluated against the merge of the sources A and B, returns the answer in figure 5

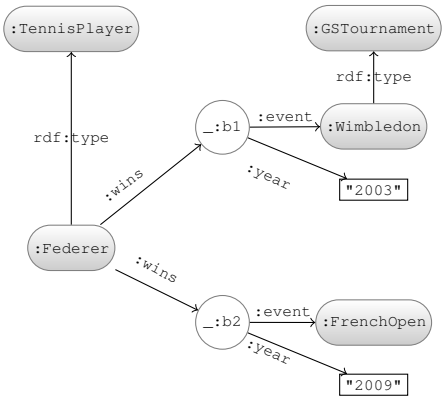

Fig. 1. RDF source A

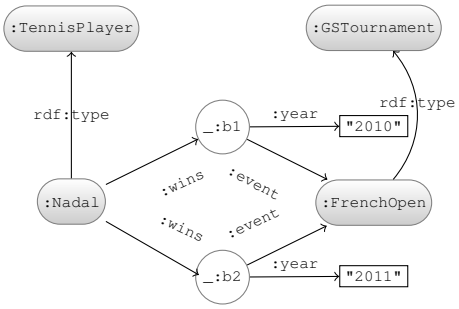

Fig. 2. RDF source B

This answer set strictly includes that which is obtained by evaluating the query in figure 4 against each source separately before taking the union of the result, viz. figure 6. In other words, the sum of the whole is bigger than the sum of the parts. Specifically, 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.

A distributed query processor should, in the absence of a good reason for sacrificing completeness, query

[1] It is a slightly modifed version of an example from [20].

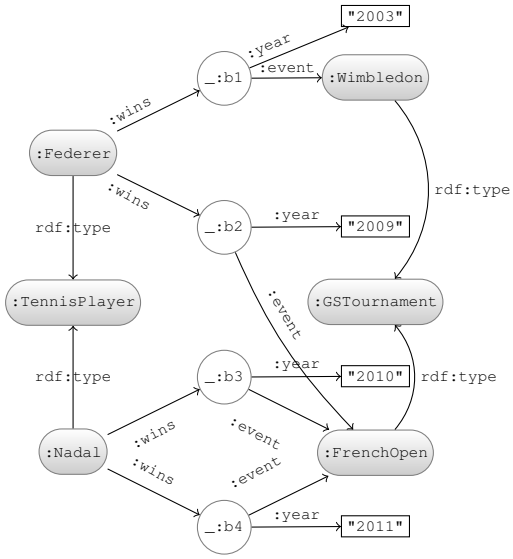

Fig. 3. The union of sources A and B modulo renaming of blank nodes.

```
SELECT ?name ?year WHERE {
  ?athl a :TennisPlayer.
  ?athl :wins ?x.
  ?x :event ?event.
  ?x :year ?year.
  ?event a :GSTournament. }
```

Fig. 4. Get Grand Slam triumphs

name	year
“Roger Federer"	2003
“Roger Federer"	2009
“Rafael Nadal"	2010
“Rafael Nadal"	2011

Fig. 5. Answer over the merge of A and B

name	year
“Roger Federer"	2003
“Rafael Nadal"	2010
“Rafael Nadal"	2011

Fig. 6. Union of answers over A and B

the whole. That is, by combining results from partial queries evaluated separately against source A and B, it should return the table in Figure 5 as the answer to query in Figure 4.

Generalizing from the present example one thus arrives at the following intuitive criterion of adequacy for a distributed query processor: the same answer should be returned as that which would be obtained were the query to be evaluated over the merge of the contributing sources. This concept of adequacy will have to be made more precise later, for there is leeway for different interpretations of it due to the appeal to the 'sameness' of answer sets. For now, however, and in relation to the example at hand, the question becomes: how must the query in figure 4 be decomposed and processed (vagueness intended) in order to satisfy the adequacy criterion?

Given the obviousness of all that has been said so far, it is somewhat surprising that there is no simple answer to this question. This is due to the fact that source A and B—as recommended by the Semantic Web Best Practices and Deployment Working Group—use blank nodes for expressing three-place predicates, in this case the predicate “X wins Y in year Z”. Therefore, the information expressed by the graphs depends on joins on 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 we shall see, this has to do with anaphoric reference being lost whenever the same blank node is processed in two separate execution contexts.

It is interesting to register that none of the more straightforward and better known query-decomposition strategies from the literature will work for this example. Consider for instance the *even decomposition*, so called in [39] (cf. Section 6) as implemented in DARQ [33]. This is the decomposition that evaluates 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 `?x :event ?event` and `?x :year ?year` from the query in figure 4 separately against each of A and B. Collecting the solutions, say, from source source A in separate tables we have the answer sets in figures 7 and 8.

Here, 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. This is mandatory according to SPARQL 1.1. specification which requires every distinct query to be treated as a separate query execution context. More precisely, every query defines

?x	?event
_:$b1_c$	:Wimbledon
_:$b2_c$	:FrenchOpen

Fig. 7. Answers to `?x :event ?event` over A

?x	?year
_:$b1_d$	2003
_:$b2_d$	2009

Fig. 8. Answers to `?x :year ?year` over A.

a distinct and sealed scope for blank node identifiers, which means that a blank node from one execution context cannot be referenced in another. From a logical point of view this is an entirely legitimate restriction. Blank nodes are similar to existential variables and existential variables are anaphors within the same quantificational context only. Nevertheless, it is detrimental for the even distribution, because join information is not preserved by the two tables. In sum, the even distribution is not a decomposition strategy that can be relied on to remain faithful to the 'sum total' of information contained in the contributing sources.

The problem generalizes, for no other well-known decomposition strategy fares any better. Take the one implemented in FedX [38] for instance. The FedX decomposition is similar to DARQ except that it groups triple patterns that are satisfiable by a single source only—so called *exclusive groups*—source into the same sub-query. This strategy is studied under the name of the *standard decomposition* in [39].

Exclusive groups anchor the triples involved to the same execution context, so they may in fortuitous cases solve the problem. However, the example currently under consideration shows that this is not in general so. Indeed, in the cases where it *is* so, it comes down to luck, because exclusivity has nothing to do essentially with blank nodes. In particular, there are no exclusive groups in the query in figure 4 wrt. source A and B, so the even and standard decompositions are the same in this case.

As a final example, consider the decomposition strategy implemented in ADERIS [26]. ADERIS evaluates the maximal sub-pattern of the global pattern that is potentially answerable by a source against that source and before taking the union of the results. In the present example, all of the RDF properties in the query occur in both of the sources A and B. Hence, the whole query is evaluated against each of the sources, and the final result is the union of the respective answer sets. But, as we have seen, this produces the table in figure 6 which misses the cross-site join that relates Roger Federer to his 2009 win.

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 to 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. From a semantical point of view, distributed query answering is essentially about balancing these two opposing forces.

The delicateness of this balancing act has not been fully appreciated yet it seems: while it is reasonable to expect a distributed query processor to do better than to just collect solutions to the global query from each of the sources, it may so happen that the very attempt to do so, i.e. the attempt to harvest information by utilizing cross-site joins, is the very thing that causes it to do worse, because if there are joins on blank nodes in a contributing source, then that source will leak solutions if the global query is decomposed too finely.

As for the question of whether there even is a decomposition that can be made to work, the answer is affirmative for the example at hand, viz. the decomposition below:

$$P_1 := \{\texttt{?x :event ?event.}, \texttt{?x :year ?year}\}$$
$$P_2 := \{\texttt{?athl a :TennisPlayer.}\}$$
$$P_3 := \{\texttt{?athl :wins ?x.}\}$$
$$P_4 := \{\texttt{?event a :GSTournament.}\}$$

The crucial thing about this decomposition is first, that it groups together those triple patterns that match a join on a blank node, thus ensuring that the join arguments are kept within one and the same execution context so as not to lose anaphoric reference (P_1). Secondly, it is also of crucial importance that all other triples are shipped as singletons, or else cross-site joins would be lost ($P_2 - P_4$). It is not difficult to show that the procedure that takes the union $[\![P_i]\!]_A \cup [\![P_i]\!]_B$ for each $i \in [1, 4]$ before folding the results into a single answer set by joins, produces the outcome in figure 5. Yet, this solution is specific to the problem at hand and depends on detailed knowledge about the shape of the graphs being queried. For instance, one needs to know that the variable `?x` will be mapped to a blank

node—something that cannot, needless to say, be read off from the query alone.

We have the work cut out for us then: first it is necessary to show that irrespective of the shape of graphs, if a solution to a query is warranted by the *merge* of the contributing sources, then there is decomposition from which that solution can be reconstructed. Secondly, it must be shown that it is possible to abstract from these particular solutions a general federation scheme that yields a correct and exhaustive answer set in the absence of any knowledge about the sources, even whilst allowing for joins on blank nodes. In order to do so, however, it will be necessary to generalize the SPARQL 1.1. semantics with an extra parameter representing execution contexts—a topic for the next section.

4. Adding execution contexts to the SPARQL semantics.

The formal developments that follow are based on the standard set semantics of SPARQL 1.1. (cf. [5]), as far as it goes. "As far as it goes", because that semantics is designed for the case where a single query is evaluated against a single source in a single execution context. But what is distinctive about the distributed setting, is that the overall answer to the distributed query is assembled from partial answers to partial queries each one of which represents a distinct execution context. These contexts matter a great deal from a semantical point of view as they constitute disjoint name spaces for blank node, which in turn influences the semantics of joins.

Execution contexts are not formally defined in the SPARQL 1.1. semantics, which therefore lacks the expressive power to define distributed query processing. However, the following passage provides a fairly robust basis for a formal reconstruction: "Since SPARQL treats blank node identifiers in a results format document as scoped to the document, they cannot be understood as identifying nodes in the active graph of the dataset. If DS is the dataset of a query, pattern solutions are therefore understood to be not from the active graph of DS itself, but from an RDF graph, called the scoping graph, which is graph-equivalent to the active graph of DS but shares no blank nodes with DS or with [the query]" [32].

The scoping graph mentioned here is a purely theoretical construct—it is just another namespace for blank nodes. In other words, SPARQL distinguishes between the scopes of a query, the queried data and the result, stipulating that blank nodes cannot be shared between them. This has important formal ramifications that will be explored shortly, after the requisite semantic concepts have been introduced.

The following notational conventions will be adopted from here on: curly braces are 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 $f(A)$ will be used as a 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.

Turning now to RDF specifics, let U, B and L denote pairwise disjoint infinite sets of IRIs, blank nodes, and literals respectively. In conformity with the nomenclature of [5], IL abbreviates $I \cup L$ and T abbreviates $I \cup B \cup L$. These latter sets will be referred to collectively as ***RDF terms***, and RDF terms will be denoted individually by u_i, v_j. Here, as everywhere else, indexes will be omitted when idle. An RDF triple is an element $(u_1, u_2, u_3) \in (I \cup B) \times I \times (I \cup B \cup L)$. If it is mnemonically convenient, terms in triples may be denoted by their roles as *s*ubject, *p*redicate and *o*bject of that triple; (s, p, o). If the elements of a triple is irrelevant to the discussion, an RDF triple will be denoted a_i where the '*a*' is meant to stand for '*a*ssertion'. If x is a triple pattern or a triple then $\pi^n(x)$ is the projection of x onto its n^{th} coordinate, presuming of course $n \leq 3$. An RDF graph is a finite set of RDF triples. RDF graphs are denoted G with or without subscripts. RDF graphs will be referred to interchangeably as a *sources*, *endpoints* or *datasets*. These terms will tend to be used for different emphasis in order to make the terminology more suggestive.

Turning now to SPARQL queries, let V be an infinite set of variables. Variables are denoted by lower case letters in the range of x to z with a question mark prepended. Variables may also be counted as terms, in which case they too are denoted u_i.

Definition 4.1. A basic graph pattern (BGP) is either

1. a singleton $\{t\}$ where t is a *triple pattern* in $(IL \cup V) \times (I \cup V) \times (IL \cup V)$, or
2. a union $P_i \cup P_j$ of BGPs P_i and P_j.

Thus, t_i denotes triple patterns and P_i denotes BGPs. For any BGP P, $var(p)$ denotes the set of variables in P. In order to avoid tedious limiting cases in proofs, it will be assumed that the set of triple patterns is disjoint from the set of RDF triples, that is, a triple pattern is assumed to contain at least one variable. By definition 4.1 (1), it is also assumed not to contain blank nodes. These assumptions incur no loss of generality.

Following [5], SPARQL queries will be identified with basic graph patterns. In other words, the distinction between the syntax and the semantics of SPARQL queries will deliberately be blurred. Queries does not become syntactic entities per se but rather sets of such, that is, they are *sets* of triple patterns. This is mathematically convenient and simplifies formal developments without loss of precision. Abiteboul et al. [1] establishes precedent in this respect.

In the SPARQL 1.1. specification answers to simple conjunctive queries aka. basic graph patterns are formalized in terms of *solution mappings*, which are partial functions μ of type $V \longrightarrow T$. The relevant definition, giving the answer $[\![P]\!]_G$ to a basic graph pattern P evaluated over an RDF graph G, reads:

Definition 4.2. Let G be an RDF dataset, and P a BGP. Let $\mu(t)$ denote the triple obtained by replacing the variables in t according to μ, and let $var(t)$ denote the set of variables occurring in t. Then

1. $[\![P]\!]_G =_{df} \{\mu \mid dom(\mu) = var(t) \text{ and } \mu(t) \in G\}$, if P is a triple pattern t, or
2. $[\![P]\!]_G =_{df} [\![P_i]\!]_G \bowtie [\![P_j]\!]_G$, if $P = P_i \cup P_j$.

Here $dom(\mu)$ denotes the domain of μ and $\mu(P)$ is understood as the set of all $(\mu(u_1), \mu(u_2), \mu(u_3))$ for $(u_1, u_2, u_3) \in P$. Individual μ will be called *solutions* whereas sets $[\![P]\!]_G$ will be called *answer sets*. The join $\bowtie$ of answer sets is defined as the set of pairwise unions of *compatible* maps where μ_i and μ_j are compatible if $x \in dom(\mu_i) \cap dom(\mu_j)$ implies $\mu_i(x) = \mu_j(x)$.

This is the standard set-based definition of answer sets, but keeping the SPARQL 1.1. specification's notion of a scoping graph in mind, it is already not entirely accurate—not even for a single source and a single execution context. For if a solution $\mu(t)$ to t over G happens to contain a blank node, it follows that $\mu(t)$ is not strictly speaking an element in G. Rather, the relationship between $\mu(t)$ is a weaker one: μ being a solution to t over G means that there is an $a \in G$ such that $\mu(t)$ and a are the same *up to renaming of blank nodes*. This in turn is just to say that $\mu(t)$ and a *simply entail* each other, where the following theorem from [20] may serve as the definition of simple entailment:

Theorem 4.1 (Simple entailment)**.** *An RDF graph G_i entails an RDF graph G_j, written $G_i \vDash G_j$, iff there is an RDF homomorphism h from G_i to G_i.*

The concept of an RDF homomorphism that figures in this definition will be defined shortly. Henceforth, whenever two answer sets are said to be the same, this is intended to mean that they simply entail each other, i.e. that they are *simply equivalent*. To belabour the obvious, this is what was hinted at in section 3 when it was said that the 'sameness' of answer sets would have to be made precise.

The upshot of this is that even in the single-source case, SPARQL semantics can be seen to have an equivalential basis. It is not all bad news. On the contrary, it fits hand-in-glove with distributed query processing, for as soon as the equivalential basis of answer sets over a single source is made formally explicit it generalizes straightforwardly to answer sets obtained by combining solutions from different execution contexts.

The notational apparatus will have to be amended slightly. Starting with a countably infinite set of contexts $\mathcal{C}$ (definition pending) what is required is a family of pairwise disjoint sets of blank nodes to match. It suffices to just stipulate that $\{B_c\}_{c \in \mathcal{C}}$ is the family of sets B_c obtained by making a copy of B in which each element is indexed by the context c. To lighten the notation the union of all these sets of blank nodes, one for each context, will henceforth be denoted $\mathbb{B}$. The set ST of SPARQL terms may now be defined as

$$ST =_{df} IL \cup \mathbb{B}$$

This is to be kept distinct from T, the set of RDF terms. The latter is the stuff that RDF graphs are made of, the former is the stuff that solutions to queries are made of. The same blank node is never contained in both.

Returning now to RDF homomorphisms:

Definition 4.3. Given two RDF graphs G_i and G_j, an RDF homomorphism is a function $h : G_i \longrightarrow G_j$ such that all of the following hold:

1. $h(u) = u$ for $u \in IL$,
2. $h(u) \in T$ if $u \in \mathbb{B}$, and
3. $(h(u_1), h(u_2), h(u_3)) \in G_j$ if $(u_1, u_2, u_3) \in G_i$.

This definition differs from the standard only in that the range of functions has been extended with the new (countable) set of blank nodes. As for the concept of an execution context, viz. the following definition:

Definition 4.4 (Execution context)**.** An execution context is a bijection $c : T \longrightarrow ST$ which is such that

1. $u \in IL$ implies $c(u) = u$,
2. $u \in B$ implies $c(u) \in B_c$,

The idea now, is to let the partial map μ, as defined in the standard semantics, figure merely as the mathematical relationship between a graph pattern and a graph that is to be portrayed by solutions proper. Stated differently, μ is no longer considered a solution per se, rather a solution μ_c is a mapping μ *modulo a context* c that provides the solution with a unique set of names for its blank nodes:

Definition 4.5. Given a $\mu : V \longrightarrow T$ and a context c, $\mu_c : V \longrightarrow ST$ is defined as follows:

1. $\mu_c(u) = \mu(u)$ whenever $\mu(u) \in IL$
2. $\mu_c(u) = c(\mu(u))$ otherwise.

Generalizing definition 4.2 accordingly yields.

Definition 4.6 (Generalized evaluation)**.** Let $\mathscr{G}$ denote a set of RDF sources, then:

1. $[\![P]\!]^c_G =_{df} \{\mu_c \,|\, \mu \in [\![P]\!]_G\}$
2. $[\![P]\!]^c_G =_{df} [\![P_i]\!]^c_G \bowtie [\![P_j]\!]^c_G$, if $P = P_i \cup P_j$.
3. $[\![P]\!]^c_{\mathscr{G}} =_{df} [\![P]\!]^c_{\bigcup_{G_i \in \mathscr{G}} G_i}$.

The following pair of lemmas is unglamorous but important:

Lemma 4.2. *An execution context c is an RDF homomorphism from $\mu(P)$ to $\mu_c(P)$.*

Proof. Suppose $\mu \in [\![P]\!]_G$, and $(\mu(u_1), \mu(u_2), \mu(u_3)) \in \mu(P)$ for $(u_1, u_2, u_3) \in P$, where $u_1, u_2, u_3 \in IL \cup V$. Then, by definition 4.6 (1) $\mu_c \in [\![P]\!]^c_G$ i.e. $(\mu_c(u_1), \mu_c(u_2), \mu_c(u_3)) \in \mu_c(P)$. By the RDF semantics, $\mu(u_2) \in I$ from which it follows that $\mu(u_2) = \mu_c(u_2)$ by definition 4.5 (1), hence $\mu_c(u_2) \in I$. Now, assume wlog. that $\mu(u_1), \mu(u_3) \in B$, as the case for $\mu(u_1), \mu(u_3) \in IL$ would proceed as for u_2 above.

By definition 4.5 (1) and (2), we then have $(c(\mu(u_1)), \mu(u_2), c(\mu(u_3))) \in \mu_c(P)$, and by definition 4.4 (1) we have $c(\mu(u_2)) = \mu(u_2)$. Therefore $(c(\mu(u_1)), c(\mu(u_2)), c(\mu(u_3))) \in \mu_c(P)$ and hence c satisfies definition 4.3 (3). That c also satisfies definition 4.3 (1) and (2) is immediate from definition 4.4. □

Lemma 4.3. *The converse of a context c is an RDF homomorphism.*

Proof. By Lemma 4.2, c is a homomorphism which by definition 4.4 is bijective. Hence c^{-1} is a graph homomorphism. To verify that it is in fact an RDF homomorphism it suffices to note that definition 4.3 (1) imposes the same restriction as definition 4.4 (1) and that definition 4.4 (2) is a stronger requirement than definition 4.3 (2). □

Taking stock so far, there is no longer such a thing as *the* set of answers to a BGP P over a graph G, strictly speaking. When blank nodes are involved there are infinitely many distinct such sets, one for each context c. These sets are all simply equivalent, though, which just means that they answer the query in essentially the same way:

Corollary 4.4. *If $var(P) \subseteq dom(\mu)$ then $\mu(P)$, $\mu_c(P)$ and $\mu_d(P)$ are all simply equivalent irrespective of the choice of c and d.*

Proof. By Lemma 4.2 and 4.3 c is an RDF homomorphism from $\mu(P)$ to $\mu_c(P)$ and c^{-1} is an RDF homomorphism in the converse direction. By Theorem 4.1 then $\mu(P) \dashv\vdash \mu_c(P)$. By similar reasoning $\mu(P) \dashv\vdash \mu_d(P)$, and by the property of euclideanness for equivalence relations $\mu_c(P) \dashv\vdash \mu_d(P)$. □

Turning now to *distributed* query answering and its semantics, its formalization is a matter of generalizing the $\bowtie$ relation to apply to answer sets from different sources and parametrized by different execution contexts:

$$[\![P_i]\!]^c_G \bowtie [\![P_j]\!]^d_H$$

Such expressions will henceforth be stipulated to denote the set of unions $\mu_c \cup \mu'_d$ of pairs of compatible maps $\mu_c \in [\![P_i]\!]^c_G$ and $\mu'_d \in [\![P_j]\!]^d_H$. There are a few things to note:

Theorem 4.5. *There is in general no context e such that $[\![P_i]\!]^c_G \bowtie [\![P_j]\!]^d_H = [\![P_i \cup P_j]\!]^e_{G \cup H}$.*

Proof. It suffices to argue the case where $G = H$. Suppose for reduction that there is such a context e for every $P := P_i \cup P_j$ and every source G. Then it is possible to choose P, G, μ_c and μ_d in such a way that $\mu_e = \mu_c \cup \mu_d$, where $\mu_c \in [\![P_i]\!]^c_G$ and $\mu_d \in [\![P_j]\!]^d_G$, although $ran(\mu_c) \cap B_c \neq \emptyset$ and $ran(\mu_d) \cap B_d \neq \emptyset$.

But then $ran(\mu_e) \cap B_c \neq \emptyset$ and $ran(\mu_e) \cap B_d \neq \emptyset$, either of which contradicts clause (2) of the definition 4.4 of contexts. □

What this means is that there is in general no algebra on contexts and sources that allows one to reduce $[\![P_1]\!]^c_G \bowtie [\![P_2]\!]^d_G$ to a single execution context. In particular we do not have $[\![P]\!]^c_G \bowtie [\![P]\!]^d_G = [\![P]\!]^{c\cup d}_G$ or anything of the sort. This is as it should be. Distributed queries do not run in a single execution context. Theorem 4.5 simply reflects this fact.

As a consequence of this, there are two different things that need to be recognized as solutions to a query: first, any solution to the query over a single source, and secondly, any solution obtained by joining partial answers from *different* sources or contexts. Call the latter *intercontextual joins*, and let them be denoted with (possibly subscripted) ρ. Intercontextual joins generalizes cross-site joins: whereas a cross-site join is an intercontextual join, the converse does not necessarily hold. The ρ notation will be used primarily as a convention for indicating in the relevant parts of a proof or line of reasoning that an appeal to a context is neither required nor apt. We have:

Theorem 4.6. *Let G_m and G_n be two RDF graphs that are standardized apart. If $\rho \in [\![P_i]\!]^c_{G_m} \bowtie [\![P_j]\!]^d_{G_n}$ then there is a context e such that $\mu_e \in [\![P_i \cup P_j]\!]^e_{G_m \cup G_n}$ and $\mu_e(P_i \cup P_j) \dashv\vdash \rho(P_m \cup P_n)$.*

Proof. By definition 4.6 (2) it suffices to show the property for a μ_e in $[\![P_i]\!]^e_{G_m\cup G_n} \bowtie [\![P_j]\!]^e_{G_m\cup G_n}$. So suppose $\rho \in [\![P_i]\!]^c_{G_m} \bowtie [\![P_j]\!]^d_{G_n}$. Then $\rho \in [\![P_i]\!]^c_{G_m\cup G_n} \bowtie [\![P_j]\!]^d_{G_m\cup G_n}$ whence $\rho = \mu_c \cup \mu'_d$ for μ_c in the first and μ'_d in the second of the join arguments. By Corollary 4.4 $\mu_c(P_i) \dashv\vdash \mu_e(P_i)$ for an arbitrarily chosen e, and $\mu'_d(P_j) \dashv\vdash \mu'_e(P_j)$ for the same e. From the former it follows by Theorem 4.1 that there is a pair (h_1, g_1) of RDF homomorphisms such that h_1 maps $\mu_c(P_i)$ into $\mu_e(P_i)$ and g_1 maps $\mu_e(P_i)$ back into $\mu_c(P_i)$. From the latter it follows that there is a pair (h_2, g_2) linking $\mu'_d(P_j)$ and $\mu'_e(P_j)$ in the same way. Thus, since $\rho = \mu_c \cup \mu'_d$ entails $\rho(P_i \cup P_j) = \mu_c(P_i) \cup \mu'_d(P_j)$, it only remains to show that A) $h_1 \cup h_2$ is an RDF homomorphism from $\mu_c(P_i) \cup \mu'_d(P_j)$ to $\mu_e(P_i) \cup \mu_e(P_j)$ and that B) $g_1 \cup g_2$ is a homorphism in the converse direction.

For A) it suffices to show that $h_1 \cup h_2$ is well-defined, i.e. that h_1 and h_2 are compatible. So, suppose for reduction that there is a $t \in dom(h_1) \cap dom(h_2)$ such that $h_1(t) \neq h_2(t)$. By the definition of an RDF homorphism t must be a blank node. However, the contexts c and d, by definition 4.4, have disjoint ranges when restricted to blank nodes, so $t \notin dom(h_1) \cap dom(h_2)$ contrary to assumption.

for B) the reasoning is entirely similar to A) so suppose for reduction that there is a $t \in dom(g_1) \cap dom(g_2)$ such that $g_1(t) \neq g_2(t)$. Then again t must be a blank node, by the definition of an RDF homomorphism. However G_m and G_n have been assumed to be standardized apart so $t \notin dom(g_1) \cap dom(g_2)$ contrary to assumption.

□

As this argument can obviously be extended to all finite unions of BGPs by a straightforward induction, it is in effect akin to a soundness result: as long as execution contexts do not share blank node identifiers with one another or with RDF graphs, and as long as the scope of RDF graphs are treated as mutually exclusive, distributed query processing, understood as assembling intercontextual joins across sources and contexts, never produces solutions that are not warranted by the union of those sources.

5. There is always a solution.

To recapitulate briefly, the point of introducing an explicit parameter for execution contexts is to be able to reason about the total information content held by a set of answers each of which was generated by a different query over a distinct source—the total information content being given by the join over all these sets. Now, as each query will set up a separate scope for blank node identifiers that is disjoint from all others, answer sets cannot in the distributed case be joined freely. In particular, any join on blank nodes, if it spans two or more execution contexts, must be deemed a false positive. Conversely, any join on blank nodes that refers to a single context is ok. In other words, execution contexts is essentially a way to keep tabs on legitimate and illegitimate joins.

The present section is concerned with the first of the tasks that were defined towards the end of section 3 which, having introduced execution contexts into the SPARQL semantics, it is now possible to address: the aim is to show that irrespective of the shape of graphs, if a solution to a query is warranted by the *merge* of the contributing sources, then there is decomposition from which that solution can be reconstructed.

The investigative strategy that is adopted here is to reverse-engineer the problem by showing that every solution over the merge of the contributing sources *induces* a decomposition that fits the bill. More specifically, the idea is to let those parts of the given solution that have joins on blank nodes, and blank nodes only, dictate how the corresponding query pattern is to be decomposed; for if a graph pattern is connected by blank nodes only, it must derive from a single source, so the sub-query that matches it must be grouped and evaluated in a single execution context. A definition of connectedness that covers both RDF graphs and SPARQL query patterns will be required:

Definition 5.1. Let d, e and f be triples over any set-theoretic universe U. There is a path from d to f if:

1. d and f share an element, or
2. there are paths from d to e, and from e to f.

A set $S \subseteq U^3$ is connected, if there is a path between every pair of elements of S.

Next, we shall be interested in those subpatterns P_i of a global query P that selects connected subgraphs of the source being queried. These subgraphs are further required to only have blank nodes in join position, and to be maximal in this respect. If P_i selects a subgraph matching this description, then it will be called a b-*component* of P:

Definition 5.2. Let μ_g be any solution in $[\![P]\!]^g_{\mathscr{G}}$ and let $P_i \subseteq P$. Then P_i is a b-component of P iff

1. $\mu_g(P_i)$ is connected.
2. If $a_m, a_n \in \mu_g(P_i)$ and $\pi^i(a_m) = u = \pi^j(a_n)$ for some $i, j \in \{1, 3\}$ then $u \in B$.
3. P_i is maximal wrt. 1 and 2.

In the limiting case, definition 5.2 makes every triple pattern that does not, modulo μ_g, select an assertion with blank nodes in it a b-component. This is an important case to keep track of in the following.

Example 5.1. *Consider the merge of the RDF graphs A and B in figure 3 and the query in figure 4. Let μ_g be the solution that assigns*

$?\texttt{athl} \mapsto$ *"Federer"*

$?\texttt{x} \mapsto \_ : b_g$

$?\texttt{event} \mapsto :\texttt{FrenchOpen}$

$?\texttt{year} \mapsto$ *"2009"*

Then the b-components of the query, relative to μ_g, are given by the decomposition in section 3, namely

$P_1 := \{$`?x :event ?event.`, `?x :year ?year`$\}$
$P_2 := \{$`?athl a :TennisPlayer.`$\}$
$P_3 := \{$`?athl :wins ?x.`$\}$
$P_4 := \{$`?event a :GSTournament.`$\}$

Although this example seems to indicate otherwise, it is a fact of some importance that $\mu_g(P)$ being a b-component does *not* entail that P is connected. Consider the following example:

Example 5.2. *Put $G := \{(b, p_1, o_1), (b, p_2, o_2)\}$ where b is assumed to be a blank node. Let $P := \{(?x, p_1, o_1), (?y, p_2, o_2)\}$. Then any $\mu_g \in [\![P]\!]^g_G$ maps both $?x$ and $?y$ to b. Since G is connected, and maximally so, $\mu_g(P)$ is consequently a b-component. Yet, P itself is not connected.*

The significance of this is that one cannot limit ones' view to connected subqueries when searching for a complete federation scheme. There will be more to say about this in the next section.

It will be convenient to have a notation that assembles all the b-components that a particular solution induces:

Definition 5.3. The set of b-components of P relative to μ_g is picked out by the function $f(\mu_g, P)$.

The next two lemmas record some important properties about f as so defined.

Lemma 5.1. *$f(\mu_g, P)$ partitions P.*

Proof. According to definition 5.2 the empty set is not a b-component, hence any $P_i \in f(\mu_g, P)$ is non-empty. For exhaustiveness we need to show that

$$P = \bigcup_{P_i \in f(\mu_g, P)} P_i$$

The right-to-left inclusion is trivial. For the converse suppose $t \in P$. If t is a b-component then $\{t\} \in f(\mu_g, P)$, so there is nothing to prove. If not, then, since $\mu_g(t)$ satisfies definition 5.2 (1) and (2) (the second vacuously), it must violate definition 5.2 (3). I. e. $\{t\}$ is not maximal wrt. definition 5.2 (1) and (2). Hence, $\{t\}$ can be expanded to a set $P_i \subseteq P$ such that $P_i \in f(\mu_g, P)$, which completes the verification of exhaustiveness.

It remains to show that the elements of $f(\mu_g, P)$ are pairwise disjoint. So, let P_i, P_j be an arbitrarily chosen

pair of elements of $f(\mu_g, P)$ and assume for the sake of contradiction that $a \in P_i \cap P_j$ for $P_i \neq P_j$. There are two cases to consider:

Case 1. a contains no blank nodes: Then, by definition 5.2 the singleton set $\{a\}$ is a b-component. Since b-components are maximal it follows that $P_i = \{a\} = P_j$ and therefore $P_i = P_j$ contrary to assumption.

Case 2. a contains a blank node.: There are two subcases: either $P_i \subset P_j$, or $P_i \not\subseteq P_j$.

Case 2.1. $P_i \subset P_j$: Then P_i is not maximal as per definition 5.2(3), hence not a b-component (contradiction).

Case 2.2. $P_i \not\subseteq P_i$: Since they are both b-components, they each are connected only on blank nodes. However, since they overlap on a, this means that $P_i \cup P_j$ is also a b-component contradicting the maximality of either of P_i and P_j.

□

Lemma 5.2. *f is a function.*

Proof. Let $x := (\mu_g, P)$ and suppose $(x, Y_i), (x, Y_j) \in f$, we need to show that $Y_i = Y_j$. Suppose for reduction that $Y_i \neq Y_j$, say wlog. $P' \in Y_i \setminus Y_j$. Since $P' \in Y_i$ it is a b-component of P relative to μ_g. By Lemma 5.1, Y_j is a partition of P, so $P' \subseteq \bigcup Y_j$. However, $P' \notin Y_j$, thus there are two cases to consider:

Case 1. P' is a proper subset of a cell in Y_j, that is, $P' \subset P'' \in Y_j$. Then contrary to assumption P' is not a b-component.

Case 2. P' is split among cells in Y_j, that is, there exists a $P'' \in Y_j$ s.t. $P' \not\subseteq P'' \in Y_j$ but $P' \cap P'' \neq \emptyset$. Since P' and P'' are b-components, they each satisfy 5.2 (1) and (2). But since $P' \cap P'' \neq \emptyset$ this is true also of $P' \cup P''$ contradicting the maximality of P' and P'' in this respect.

□

The combined significance of these two lemmas may be explained as follows: Lemma 5.2 fixes a single set $f(\mu_g, P)$ of sub-patterns of P, which according to Lemma 5.1 mutually exhausts P. Taken together, therefore, they show that talk of $f(\mu_g, P)$ as *one decomposition* of P is legitimate. This decomposition has special properties:

Lemma 5.3. *Let P be a query, $\mathscr{G} = \bigcup_{i \in I} G_i$ a set of sources that are standardized apart, and $\mu_g \in [\![P]\!]^g_{\mathscr{G}}$. Then, for any $P_i \in f(\mu_g, P)$ there exists a source $G_k \in \mathscr{G}$ such that $\mu'_c \in [\![P_i]\!]^c_{G_k}$ and $\mu_g(P_i) \dashv\vdash \mu'_c(P_i)$ for any execution context c.*

Proof. Put $\mu' := (\mu_{|vars(P_i)})$. Then $\mu' \in [\![P_i]\!]_{\mathscr{G}}$. By Corollary 4.4 it suffices to show that $\mu' \in [\![P_i]\!]_{G_k}$ for some $G_k \in \mathscr{G}$. The proof subdivides into two cases:

Case 1. $\mu'(P_i)$ contains blank nodes: Suppose for reduction that there is no single $G_k \in \mathscr{G}$ with $\mu' \in [\![P_i]\!]_{G_k}$. Then $\mu' \in [\![P_i]\!]_{\mathscr{G}}$ means that $\mu'(P_i)$ must span at least two sources $G_k, G_j \in \mathscr{G}$. Now, $\mu'(P_i)$ is connected, since $P_i \in f(\mu_g, P)$, so it is possible to choose triple patterns $t_m, t_n \in P_i$ such that $a_m := \mu'(t_m) \in G_j$, $a_n := \mu'(t_n) \in G_k$ and such that $\{a_m, a_n\}$ is connected. By definition 5.2 (2) a_m and a_n share a blank node. However, this contradicts the assumption that G_k and G_j are standardized apart.

Case 2. $\mu_g(P_i)$ does not contain a blank node: Then P_i is a singleton $\{a\}$, so the desired result follows immediately.

□

In words, Lemma 5.3 demonstrates that every element P_i of $f(\mu_g, P)$ is such that there exists a *single* source in the set of sources in question capable of answering P_i in a manner equivalent to μ_g. Needless to say, therein lies the essence of federation, or more accurately, one half of it. The other "half" is given by Lemma 5.4:

Lemma 5.4. *Suppose $P_m, P_n \in f(\mu_g, P)$, $\mu'_c(P_m) \dashv\vdash \mu_g(P_m)$ and $\mu''_d(P_n) \dashv\vdash \mu_g(P_n)$. Then $\mu'_c(P_m) \cup \mu''_d(P_n) \dashv\vdash \mu_g(P_m \cup P_n)$.*

Proof. Suppose the conditions of the theorem hold. Since $\mu_g(P_m \cup P_n) = \mu_g(P_m) \cup \mu_g(P_n)$ it suffices to show $\mu'(P_m) \cup \mu''_d(P_n) \dashv\vdash \mu_g(P_m) \cup \mu_g(P_n)$. By Theorem 4.1, this in turn reduces to demonstrating the existence of an RDF homorphism from left to right and one from right to left. By Lemma 5.3 the theorem holds for each of the component patterns. That is, there is a pair of RDF homomorphisms (h_1, g_1) of $\mu'_c(P_m)$ to $\mu_g(P_m)$ and back, and a pair (h_2, g_2) of $\mu''_d(P_n)$ to $\mu_g(P_n)$ and back. We show that $(h_1 \cup h_2, g_1 \cup g_2)$ is a pair of RDF homomorphisms of $\mu'_c(P_m) \cup \mu''_d(P_n)$ to $\mu_g(P_m) \cup \mu_g(P_n)$ and back.

For $h_1 \cup h_2$, it is necessary to verify first of all that it is well-defined, i.e. that for any $u, v \in ST$ if $u = v$ then $h_1(u) = h_2(v)$. The verification subdivides into two cases depending on the sets to which u and v belong. To ease the notation put $h^+ := h_1 \cup h_2$.

Case 1. $u, v \in \mathbb{B}$. If $u, v \in \mu'_c(P_m)$ then $h^+(u) = h_1(u) = h_1(v) = h^+(v)$. Similarly, if $u, v \in \mu''_d(P_n)$ then $h^+(u) = h_2(u) = h_2(v) = h^+(v)$. Suppose therefore wlog. that $u \in \mu'_c(P_m)$ and $v \in \mu''_d(P_n)$.

Then it follows by definition 4.4 (2) that $u \neq v$ so well-definedness holds vacuously.

Case 2. $u, v \in IL$: Then since h_1 and h_2 are RDF homomorphisms, they are the identity on u and v, whence

$$\begin{aligned} h^+(u) &= (h_1 \cup h_2)(u) \\ &= h_1(u) \cup h_2(u) \\ &= u \\ &= v \\ &= h_1(v) \cup h_2(v) \\ &= (h_1 \cup h_2)(v) \\ &= h^+(v) \end{aligned}$$

It remains to verify that h^+ is an RDF homomorphism, so suppose $(s, p, o) \in \mu'_c(P_m) \cup \mu''_d(P_n)$. Then either $(s, p, o) \in \mu'_c(P_m)$ or $(s, p, o) \in \mu''_d(P_n)$. Hence, either $h^+((s, p, o)) = h_1((s, p, o))$ or either $h^+((s, p, o)) = h_2((s, p, o))$. In both cases h^+ is an RDF homomoprhism, since both of h_1 and h_2 are. This completes the case for $h_1 \cup h_2$.

For $g_1 \cup g_2$, put $g^+ := g_1 \cup g_2$. To verify that g^+ is well-defined it suffices to check the case where $u, v \in \mathbb{B}$, the other one being similar to that for h^+.

Suppose that either $u, v \in \mu_g(P_m)$ or $u, v \in \mu_g(P_n)$ holds. In the former case $h^+ = h_1$ and in the latter $h^+ = h_2$. The desired result follows immediately from either by the functionality of h_1 and h_2. It suffices, therefore, to confirm that $\mu_g(P_m)$ and $\mu_g(P_n)$ do not share blank nodes.

Suppose for contradiction and without loss of generality that $(b_g, p_i, o_i) \in \mu_g(P_m)$ and $(b_g, p_j, o_j) \in \mu_g(P_n)$. Then $\mu_g(P_m) \cup (b_g, p_j, o_j)$ remains connected and connected by blank nodes only, that is it satisfies definition 5.2(1) and (2). Choose $t \in P_n$ such that $(b_g, p_j, o_j) = \mu_g(t)$. Then $\mu_g(P_m \cup t) = \mu_g(P_m) \cup (b_g, p_j, o_j)$. Since $f(\mu_g, P)$ is a partition we have $P_m \subset P_m \cup t$, hence P_m is not maximal wrt. definition 5.2(1) and (2), contradicting $P_m \in f(\mu_g, P)$. This completes the verification that g^+ is well-defined.

The verification that g^+ is an RDF homomorphism is exactly like that for h^+, so the proof is complete. □

It may be worthwhile to pause to register what it is that makes this proof work: it is essentially a question of building larger RDF homomorphisms from smaller ones. This requires certain simple conditions to be met, most importantly that the union of functions be a function. Since the smaller functions in question are RDF homomorphisms, and RDF homomorphisms by definition map URIs and literals to themselves, the only way this could go wrong is if the mapped element were a blank node. However, this cannot happen for the left to right direction, since execution contexts have disjoint ranges, and it cannot happen for the right to left direction since the solutions in question do not share blank nodes. Thus the theorem flows from two things: the relativization of answer sets to execution contexts and the splitting of queries into b-components.

It is now possible to prove the converse of Theorem 4.6 and the main result of this section:

Corollary 5.5. *Let $\mathscr{G} := \bigcup_{i \in I} G_i$ be a set of sources, standardized apart, and let $\mu_g \in [\![P]\!]^g_{\mathscr{G}}$. Then there is a partition $P_1, \ldots, P_n$ of P such that, $\rho \in [\![P_m]\!]^{c_1}_{G_1} \bowtie \ldots \bowtie [\![P_n]\!]^{c_n}_{G_n}$ for arbitrarily chosen $c_1, \ldots, c_n$ and $\mu_g(P) \dashv\vdash \rho(P)$.*

Proof. Suppose the conditions of the corollary hold. Put $P_1, \ldots, P_n = f(\mu_g, P)$. By Lemma 5.1 we have $P = P_1 \cup \ldots \cup P_n$. Lemma 5.3 entails that for every P_k there exists a source, say G_k, a context, say c_k, and a solution mapping, say μ^k, such that $\mu^k_{c_k} \in [\![P_k]\!]^{c_k}_{G_k}$ and $\mu^k_{c_k}(P_k) \dashv\vdash \mu_g(P_k)$. By extension of Lemma 5.4 to finite unions it follows that $\mu^1_{c_1}(P_1) \cup \ldots \cup \mu^n_{c_n}(P_n) \dashv\vdash \mu_g(P_1) \cup \ldots \cup \mu_g(P_n) = \mu_g(P)$. Therefore, selecting $\rho := \mu^1_{c_1} \bowtie, \ldots, \bowtie \mu^n_{c_n}$ proves the corollary. □

Whereas Theorem 4.6 says that cross-site joins can never be incorrect, provided that tabs are kept on execution contexts, Corollary 5.5 states that every solution to a query P returned by the merge of the contributing sources is contained in the join of the separate evaluation of the cells of *some* decomposition of P. Thus there is always a solution distributed in the contributed sources if there is a solution in their union modulo renaming of blank nodes—i.e. in their merge.

What Corollary 5.5 does not say, is how to obtain this decomposition when we do not know the structure of the sources. It was only possible to prove that a decomposition exists by assuming prior knowledge of a solution that can be used to partition the query. But of course, such a solution cannot be assumed to be available if federation is to produce information one does not already possess. It remains, therefore, to abstract from Theorems 4.6 and 5.5 a general federation scheme that will deliver all solution, and only solu-

tions, over any set of contributing sources in the absence of any knowledge about them. For this, we shall need the concept of a federation scheme.

6. Federation schemes

The concept of a federation scheme was first introduced in [39] as a mathematical abstraction for reasoning about the distributed query answering process. It is meant to highlight the two complementary aspects of this process, which may be taken to consists respectively in decomposing a query over a set of sources and in assembling an answer to it from the partial answers returned by each contributing source. A federation scheme is thus a pair $(\mathbb{E}, \delta)$ of a *decomposition function* δ and an *evaluation rule* $\mathbb{E}$. Intuitively a decomposition function is responsible for assigning subqueries to contributing sources, whereas an evaluation rule is responsible for applying joins and unions (or in general any operation on answer-sets) in the right measure and order.

The soundness and completeness of a federation scheme is a relationship that holds wrt. a set of sets of contributing sources—or, in a more suggestive terminology, a set of *selections* of contributing sources: the federation scheme specifies how a graph pattern is to be distributed and evaluated over the selection, the set of selections defines the permissible ways of choosing sources over which to distribute the decomposed query. Stated differently, each selection of sources represents a constellation of datasets that is a good match for the federation scheme in question in the sense that the federation scheme will neither invent nor ignore answers when applied to that selection. Bordering on circularity, one might say that a set of selections of RDF datasets represents a *way* of choosing RDF datasets that keeps a federation scheme correct and exhaustive. The role of soundness and completeness theorems is thus to act as chopsticks (borrowing a metaphor from Makinson [27]) to pin down a federation scheme and a set of selections that is perfectly matched.

The present section recalls the essentials of this formal framework, and generalizes it to the new equivalence-based semantics for distributed queries as set out in the previous section.

The minimal amount of knowledge one needs about a set of sources in order to distribute a query over them, is to know their signatures. Here a signature is understood essentially as the set of concrete predicates of an RDF graph or of a SPARQL pattern. The following definition covers both cases:

Definition 6.1. Let S be a BGP or a set of RDF triples. The signature of S written $sig(S)$ is defined as

$$sig(S) =_{df} S^2 \cap I$$

where S^2 denotes the projection of S onto its second coordinates.

Note that variables and blank nodes do not add to the signature of a graph pattern, and that blank nodes do not add to the signature of an RDF graph. Using signatures to route subqueries is a common stratagem in the literature (cf. [13] and [38]).

Definition 6.2. If P is non-empty, then

$$\Sigma_{\mathscr{G}}(P) =_{df} \{G \in \mathscr{G} : sig(P) \subseteq sig(G)\}$$

Otherwise $\Sigma_{\mathscr{G}}(P) =_{df} \emptyset$.

It is a simple consequence of definitions 6.1 and 6.2 that if $a^2 \in V \cup B$ for some triple pattern a, that is, if a has a variable or blank node in predicate position, then $\Sigma_{\mathscr{G}}(a) = \mathscr{G}$ for any $\mathscr{G}$. A consequence of this is that a triple pattern that has a blank node or variable in predicate position will be routed to all datasets in a selection.

Definition 6.3 (Decomposition function)**.** A decomposition function is a binary function δ that takes a set of RDF datasets $\mathscr{G}$ and a BGP P and returns a set of pairs $(P_i, \mathscr{G}_j)$ such $P_i \neq \emptyset$ and such that all of the following hold:

1. $\mathscr{G}_j \subseteq \Sigma_{\mathscr{G}}(P_i)$
2. if $(P_1, \mathscr{G}_i), (P_2, \mathscr{G}_j) \in \delta(\mathscr{G}, P)$ and $P_1 = P_2$ then $\mathscr{G}_i = \mathscr{G}_j$
3. $\bigcup_{(P_i, \mathscr{G}_j) \in \delta(\mathscr{G}, P)} P_i = P$

Clause (1) prevents a decomposition function from behaving erratically when it comes to assigning subqueries to datasets. That is, a BGP is only assigned to datasets that can potentially answer it. Clause (2) ensures that the value of a decomposition function on any selection of datasets $\mathscr{G}$ and any BGP P is itself a function, that is, that every subquery of P is assigned to exactly one subset of $\mathscr{G}$. Finally, clause (3) expresses a necessary condition for soundness: δ distributes P over $\mathscr{G}$ only if P is decomposed into sub-queries that jointly exhaust P. Note that δ as so defined is a partial function, that is, (1) and (3) can not be satisfied for ar-

bitrary choices of $\mathscr{G}$ and P, reflecting the fact that the accumulated signature of the datasets may not cover the signature of the global query. If it does then δ will henceforth be said to be defined *at* $\mathscr{G}$ and P.

6.1. Examples of decomposition functions

Section 3 mentioned the notion of an exclusive group of a query P relative to a selection of datasets $\mathscr{G}$. This concept can be defined as :

Definition 6.4. The subpattern of BGP P that is exclusive to a dataset G, relative to a set of RDF datasets $\mathscr{G}$, is the set:

$$E_G(P) =_{df} \{t \in P : \Sigma_{\mathscr{G}}(t) = \{G\}\}$$

A triple pattern that is not an element of an exclusive group is called a non-exclusive triple pattern relative to the same selection of datasets.

In this form, definition 6.4 goes back to [38].

The following theorem, the proof of which can be found in [39], gives examples of decomposition functions:

Theorem 6.1. *Let $\mathscr{G}$ be any set of RDF datasets and P any BGP. Stipulate that $(P_i, \mathscr{G}_i) \in \delta(\mathscr{G}, P)$ iff $\mathscr{G}_i = \Sigma_{\mathscr{G}}(P_i)$ and one of the following conditions hold:*

1. *Even decomposition: P_i is a singleton $t \in P$.*
2. *Standard decomposition: either*
   a) *P_i is a non-exclusive singleton $t \in P$, or*
   b) *$P_i = E_G(P)$ for some $G \in \mathscr{G}$*
3. *Prudent decomposition: either*
   a) *P_i is a non-exclusive singleton $t \in P$, or*
   b) *P_i is a maximal connected subset of $E_G(P)$ for some $G \in \mathscr{G}$*

Then δ is a decomposition function in the sense of definition 6.3.

Example 6.1. *Consider the query in listing 1. This is query LS5 from the FedBench suite [37] which asks for all drugs from Drugbank, together with the URL of the corresponding page stored in KEGG and the URL to the image derived from ChEBI.*

The signature of the query divides among the FedBench datasets as specified in table 1. Presupposing this division, table 2 gives the even, standard and prudent decompositions respectively. Blocks in a column correspond to subqueries and are labelled with the datasets to which that subquery is assigned. For instance, the standard decomposition assigns the subquery consisting of line 2 and 3 to KEGG, since 2 and 3 form and exclusive group for it. The non-exclusive triple pattern in line 4, in contrast, is assigned to both KEGG and ChEBI.

```
SELECT ?drug ?keggUrl ?chebiImage WHERE {
   ?drug rdf:type drugbank:drugs .
   ?drug drugbank:keggCompoundId ?keggDrug .
   ?drug drugbank:genericName ?drugBankName .
   ?keggDrug bio2rdf:url ?keggUrl .
   ?chebiDrug purl:title ?drugBankName .
   ?chebiDrug bio2rdf:image ?chebiImage .
}
```

Listing 1: Query LS5 from FedBench

Table 1
Decomposition of the signature of query LS5.

Signature element/property	Endpoint
`rdf:type`	All
`drugbank:keggCompundId`	Drugbank
`drugbank:genericName`	Drugbank
`bio2rdf:url`	KEGG, ChEBI
`purl:title`	KEGG, ChEBI
`bio2rdf:image`	ChEBI

The standard- and prudent decompositions of LS5 over KEGG, ChEBI and Drugbank are identical for this particular selection of datasets. The difference between them is revealed by reducing the selection to ChEBI and Drugbank only, viz. table 3. Here, the standard decomposition assigns the subquery consisting of line 4,5 and 6 to ChEBI, since this larger set now constitutes an an exclusive group for ChEBI. Note, however, that its join-graph is not connected since none of the variables in line 4 occur in line 5 or 6. Therefore, the prudent decomposition divides $\{4, 5, 6\}$ into its two join-connected components $\{4\}$ and $\{5, 6\}$ which are both assigned to ChEBI, only this time as separate subqueries.

6.2. Evaluation rules

Turning now to the notion of an evaluation rule, it suffices to define it abstractly as a function that produces a set of answers, in the form of *intercontextual* joins (cf. the previous section), from a decomposition:

Table 2
Decomposition of LS5 over KEGG, ChEBI and Drugbank

Nr.	Triple pattern	even	standard	prudent
1	?drug rdf:type drugbank:drugs	All	All	All
2	?drug drugbank:keggCompoundId ?keggDrug	Drugbank		
3	?drug drugbank:genericName ?drugBankName	Drugbank	Drugbank	Drugbank
4	?keggDrug bio2rdf:url ?keggUrl	KEGG, ChEBI	KEGG, ChEBI	KEGG, ChEBI
5	?chebiDrug purl:title ?drugBankName	KEGG, ChEBI	KEGG, ChEBI	KEGG, ChEBI
6	?chebiDrug bio2rdf:image ?chebiImage	ChEBI	ChEBI	ChEBI

Table 3
Decompositions of LS5 over ChEBI and Drugbank

Nr.	Triple pattern	even	standard	prudent
1	?drug rdf:type drugbank:drugs	Both	Both	Both
2	?drug drugbank:keggCompoundId ?keggDrug	Drugbank		
3	?drug drugbank:genericName ?drugBankName	Drugbank	Drugbank	Drugbank
4	?keggDrug bio2rdf:url ?keggUrl	ChEBI		ChEBI
5	?chebiDrug purl:title ?drugBankName	ChEBI		
6	?chebiDrug bio2rdf:image ?chebiImage	ChEBI	ChEBI	ChEBI

Definition 6.5 (Evaluation rule)**.** An evaluation rule is a function $\mathbb{E}$ that takes any decomposition $\delta(\mathscr{G}, P)$ and returns a set of variable mappings $\rho : V \to ST$.

Obviously, this definition hides great variety. Next:

Definition 6.6. A federation scheme is a pair $(\mathbb{E}, \delta)$ consisting of an evaluation rule and a decomposition function.

The following particular evaluation rule is a generalization to a semantics parametrized by contexts of the so-called collect-and-combine rule from [39]:

Definition 6.7 (Collect-and-combine rule)**.** Put $\Delta := \delta(\mathscr{G}, P)$. If δ is not defined at $\mathscr{G}$ and P then $\mathbb{E}^c(\Delta) =_{df} \emptyset$, otherwise

$$\mathbb{E}^c(\Delta) =_{df} \bowtie \{[P_1]_{\mathscr{G}_i} : (P_1, \mathscr{G}_i) \in \Delta\}$$

where $[P]_{\mathscr{G}} =_{df} \bigcup_{G \in \mathscr{G}} [\![P]\!]^c_G$.

The collect and combine rule is relatively simple. It takes each element in a decomposition, executes it against each source that covers its signature, and collects the results. Then it forms the coordinatewise join of the resulting set. Peeking ahead to the next section, this rule turns out to be too simple for the case where blank nodes are involved, but may serve as a concrete example to aid intuition.

Note that the sets $[P]_{\mathscr{G}}$ are not themselves parameterized by context. As they will figure only as arguments to intercontextual joins, the contexts drop out of view at this point.

6.3. Completeness.

Recapitulating briefly, the properties of soundness and completeness are relations that hold between federation schemes on the one hand and sets of selections of RDF datasets on the other. The former specifies how a graph pattern is to be distributed and evaluated, the latter defines the permissible ways of selecting sources over which to distribute the decomposed query.

One will want the federation scheme in question and the selection of sources to be perfectly matched, meaning that if a global query is processed according to the scheme it will neither invent solutions nor ignore any that are warranted by the sum total of information in the contributing sources. Recalling that sameness of solutions is defined in terms of simple equivalence, this gives the following definition of the soundness and completeness of a federation scheme wrt. to a set of selection of sources:

Definition 6.8. Let $\{\mathscr{G}_i\}_{i \in I}$ be a set of selections of RDF graphs. A federation scheme $(\mathbb{E}, \delta)$ is *complete* wrt. $\{\mathscr{G}_i\}_{i \in I}$ if for every $i \in I$, $\mu_c \in [\![P]\!]^c_{\mathscr{G}_i}$ implies that there is a $\rho \in \mathbb{E}(\delta(\mathscr{G}_i, P))$ such that $\rho(P) \dashv\vdash \mu_c(P)$. It is *sound* if the converse holds.

This definition differs from that of [39] where soundness and completeness is defined in terms of set-inclusion as $\mathbb{E}(\delta(\mathscr{G}_i, P)) = [\![P]\!]^c_{\mathscr{G}_i}$. That's fine if

one disregards blank nodes, i.e. assumes that there are none, for if there are no blank nodes then contexts are idle whence answer sets do not need to be parameterized. The inclusion-based semantics of [5] will then be sufficiently expressive to give a characterisation of the semantics of distributed queries. In the opposite case, however, contexts are needed to keep tabs on the separate scopes for blank nodes. Therefore emphasis must be shifted from set-inclusion onto simple equivalence, as explained in section 4. Definition is adjusted accordingly.

Taking sets of selections of sources (that is, sets of sets of sources) as the semantic correlate of decomposition schemes facilitates the formalization of different degrees of knowledge about the structure and content of the sources that a query is distributed over. At the extreme end of this spectrum one finds the zero-knowledge case, i.e. the case where nothing is known about the tributary sources apart from their respective signatures. The zero-knowledge case, of course, corresponds to the absence of any restriction on how to make a selection of sources to distribute a query over, i.e. it corresponds to the set of all selection of sources. Stolpe [39] proves the following:

Theorem 6.2. *Suppose δ is the even-, standard or prudent decomposition. Then $(\mathbb{E}^c, \delta)$ is sound and complete wrt. the set of all selections of RDF graphs, provided none of the selected graphs contain blank nodes.*

However, Theorem 6.2 does not generalize. That is, neither the even-, standard nor prudent decomposition guarantees completeness of query answering under the collect-and-combine rule when blank nodes are allowed.

7. A zero-knowledge complete federation scheme.

Consider the following distribution function and evaluation rule:

Definition 7.1. Let $\mathscr{G}$ be a set of sources, P a query pattern, and put $\mathscr{G}_i =_{df} \Sigma_{\mathscr{G}}(P_i)$. Then

$$\delta^b(\mathscr{G}, P) =_{df} \{(P_i, \mathscr{G}_i) : P_i \subseteq P, \mathscr{G}_i \neq \emptyset\}$$

It is immediate by this definition, that if δ^b is defined at $\mathscr{G}$ and P then $\delta^b(\mathscr{G}, P)$ includes at least one partition of P. Indeed, peeking ahead to Theorem 7.1, it can be shown to contain a sufficient number of subsets of P to reconstruct any partition of P that is induced by a solution μ_g over the merge of $\mathscr{G}$. This is what matters to completeness, of course.

Table 4
Summary of notation

Nomenclature	description
tr	a triple pattern
P	a conjunctive graph pattern
G	an RDF graph
$sig(P)$	the signature/the set of predicates of P
$\Sigma_{\mathscr{G}}(P)$	the datasets in $\mathscr{G}$ whose signature cover P
$\mathscr{G}$	a set of RDF graphs
$var(P)$	the variables occurring in P
μ	partial function from variables to RDF terms
μ_c	solution mapping, blank nodes derive from c
ρ	a join across multiple execution contexts
$dom(\mu_c)$	the domain of μ_c
$[\![P]\!]^c_G$	P evaluated over G in context c
$[\![P]\!]^c_{\mathscr{G}}$	P evaluated over the union of $\mathscr{G}$ in context c
$[P]_{\mathscr{G}}$	the union of evaluating P over *each* $G \in \mathscr{G}$

Definition 7.2. Let $\mathscr{G}$ and P be as in definition 7.1. For any distribution Δ of a query pattern P let $\mathbb{P}(\Delta)$ denote the set of partitions of P contained in Δ. Then

$$\mathbb{E}^b(\Delta) =_{df} \bigcup_{\{(P_i,\mathscr{G}_i)\}_{i\in I}\in\mathbb{P}(\Delta)} \mathbb{E}^c(\{(P_i, \mathscr{G}_i)\}_{i\in I})$$

It should be evident that $\mathbb{E}^b$ is a straightforward generalization of $\mathbb{E}^c$. More precisely, $\mathbb{E}^b$ is simply the union of the results of applying $\mathbb{E}^c$ to each partition of P that the distribution function delivers. We have:

Theorem 7.1. *The federation scheme $(\mathbb{E}^b, \delta^b)$ is sound and complete wrt. the set of all selections of sources.*

Proof. For completeness, let $\mathscr{G}$ be any selection of sources and suppose $\mu_g \in [\![P]\!]^g_{\mathscr{G}}$. By the definition of $\mathbb{E}^b$ it suffices to find a set $\{(P_i, \mathscr{G}_i)\}_{i\in I} \in \mathbb{P}(\Delta)$ such that there is a $\rho \in \mathbb{E}^c(\{(P_i, \mathscr{G}_i)\}_{i\in I})$ with $\rho(P) \dashv\vdash \mu_g(P)$. Now, by Corollary 5.5 there is a partition $P_1, \ldots, P_n$ of P such that, $\rho \in [\![P_m]\!]^{c_m}_{G_m} \bowtie \ldots \bowtie [\![P_n]\!]^{c_n}_{G_n}$ for arbitrarily chosen $c_1, \ldots, c_n$, and $\mu_g(P) \dashv\vdash \rho(P)$. We need to show first of all that this partition is an element of $\mathbb{P}(\delta^d(\mathscr{G}, P))$. This is immediate, because $\rho \in [\![P_m]\!]^{c_m}_{G_m} \bowtie \ldots \bowtie [\![P_n]\!]^{c_n}_{G_n}$ and $\mu_g(P) \dashv\vdash \rho(P)$ implies that $\Sigma_{\mathscr{G}}(P_i) \neq \emptyset$ for every P_i thus satisfying definition 7.1. Now, by the definition of $\mathbb{E}^c$ we have $\mathbb{E}^c(\{(P_i, \mathscr{G}_i)\}_{i\in I}) = [P_m]_{\mathscr{G}_m} \bowtie \ldots \bowtie [P_n]_{\mathscr{G}_n}$ so it suffices to show that ρ is in

the right hand side of this equality. By definition 6.7 we have $[\![P_i]\!]^{c_i}_{G_i} \subseteq [P_i]_{\mathscr{G}_i}$ for every $i \in [m,n]$. By monotony for $\bowtie$ under set inclusion, therefore, $[\![P_m]\!]^{c_m}_{G_m} \bowtie \ldots \bowtie [\![P_n]\!]^{c_n}_{G_n} \subseteq [P_m]_{\mathscr{G}_m} \bowtie \ldots \bowtie [P_n]_{\mathscr{G}_n} = \mathbb{E}^c(\{(P_i, \mathscr{G}_i)\}_{i\in I})$, as desired.

As for the soundness direction of the proof, this is just Lemma 4.6, so the proof is complete. □

A few comments on this result are in order: Although the completeness of $(\mathbb{E}^b, \delta^b)$ is achieved, at the end of the day, by grouping together in a single sub-query those triple patterns that are needed to capture a join on a blank node, this does not entail that the sub-queries themselves are connected. Example 5.2 showed as much. Therefore, in the absence of knowledge about the structure of the data, the distribution function δ^d has to cast a very wide net indeed. $\delta^d(\mathscr{G}, P)$ thus contains one pair for every subset of P such that the signature of P in its entirety is covered by at least one source.

When one cannot make any assumptions about the structure and content of the contributing sources, there seems to be no other solution than to evaluate *all* the possible partitions of the original query that involves only cells whose signature is covered by a source. Indeed, this can easily be seen to be not only a sufficient, but also a necessary condition for completeness, for if one of these partitions is skipped it is straightforward to tailor a selection of sources that correspond to just that partition.

Alas, in the limiting case where $\mathscr{G}$ contains sources with identical signatures, the distribution $\delta^d(\mathscr{G}, P)$ is isomorphic to the powerset of P, which in turn means that the number of partitions that must be processed by the evaluation rule is the Bell number corresponding to the cardinality of P. This requires an exponential number of steps and so is not computationally tractable. There seems to be no way around this conclusion; federation over data sources containing blank nodes is inherently hard.

The obvious remedy for this is to make sure that one has some knowledge about the sources, either by imposing *requirements* on them or by *probing* them. By 'probing' is here understood an initial round of queries, typically but not necessarily SPARQL ASK queries, designed to detect patterns in the data, typically concerning the placement of blank nodes, that can be exploited for decomposing the global query. A condition that simplifies things a great deal, and that involves both regimentation and probing is the following:

Definition 7.3 (Uniformity)**.** A dataset $\mathscr{G} := \{G_i\}_{i\in I}$ is a uniform selection of sources, or just uniform, if for all patterns P it is the case that if $\mu_c \in [\![P]\!]^c_{\mathscr{G}}$ and $\mu'_d \in [\![P]\!]^d_{\mathscr{G}}$ then $f(\mu_c, P) = f(\mu'_d, P)$.

To be sure, uniformity is a lot to ask for. It is a global requirement that demands that the same type of information be encoded by the same graph pattern in all the sources where it occurs. Uniformity does not, however, limit the space of eligible patterns. That is, it does not say anything specific about how the data must look, only that an encoding pattern must be employed consistently across all contributing sources. This need not be as strict as it sounds, since it will be the case for instance if each source is individually consistent in this sense and if in addition the contributing sources all provide solutions to different sub-patterns.

From a theoretical point of view, uniformity is interesting for two reasons: first, a very limited amount of probing suffices. In fact, a single solution to the global query is enough to determine a sound and complete federation scheme. Secondly, the evaluation rule $\mathbb{E}^d$ only has to process a single partition, and thus reduces to $\mathbb{E}^c$. All this is recorded in the following simple definition and accompanying theorem:

Definition 7.4 (Induced distro)**.** Let $\mu_c \in [\![P]\!]^c_{\mathscr{G}}$. The distribution induced by μ_c is defined as the function δ_{μ_c} such that $(P_i, \mathscr{G}_k) \in \delta_{\mu_c}$ iff

1. $\mathscr{G}_k \subseteq \mathscr{G}$,
2. $\Sigma_{\mathscr{G}}(P) = \mathscr{G}_k$, and
3. $P_i \in f(\mu_c, P)$

Theorem 7.2. *Let $\mathscr{G}$ be a uniform selection of sources. Then $(\mathbb{E}^c, \delta_{\mu_c})$ is sound and complete wrt. to $\{\mathscr{G}\}$.*

Proof. Soundness is already established. For completeness, suppose $\mu'_g \in [\![P]\!]^g_{\mathscr{G}}$. We need to show that there is a $\rho \in \mathbb{E}(\delta_{\mu_c}(\mathscr{G}, P))$ such that $\rho(P) \dashv\vdash \mu'_g(P)$. Clearly there is a $\rho' \in \mathbb{E}(\delta_{\mu'_g}(\mathscr{G}, P))$ with the required properties since $\delta_{\mu'_g}$ is induced by μ'_g. But by uniformity $\mathbb{E}(\delta_{\mu_c}(\mathscr{G}, P)) = \mathbb{E}(\delta_{\mu'_g}(\mathscr{G}, P))$ so we are done. □

Summing up, one might say that Theorem 7.1 and 7.2 constitute different extremes of a combinatorial spectrum. The former gives a complete federation scheme in which the number of required partitions of a query grows exponentially in the size of the query, whereas the latter gives a federation scheme where the number of required partitions is constant with $c = 1$.

It follows that any blend of probing and regimentation will land you somewhere in-between.

Of course, with perfect knowledge about the structure of the sources, probing will not be necessary at all. A perfect knowledge-case would be one where the contributing sources could be expected to conform to a predefined schema e.g. a resource shape [34]. In such a case, distributed query processing can be done efficiently without having to sacrifice the expressive power that comes with blank nodes.

Exploring the space between Theorem 7.1 and 7.2 should provide a rich field for for research. It is left for the future.

8. Conclusion

It has been demonstrated that the presence of blank nodes in RDF data represents a problem for distributed processing of SPARQL queries. Even though the facts of the matter are fairly simple, they seem as yet to have slipped by unnoticed, and none of the usual decomposition strategies from the literature solves it.

To mend the situation, this paper has introduced a semantics for distributed query processing in which the notion of an execution context is explicit. This makes it possible to keep tabs on the naming of blank nodes across execution contexts, which in turn makes it possible to articulate a decomposition strategy that is provably sound and complete wrt. any selection of RDF sources even when blank nodes are allowed. Alas, this strategy is not computationally tractable. However, there are ways of utilizing knowledge about the sources, if one has it, that will help considerably.

References

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

[2] M. Acosta and M.-E. Vidal. Networks of linked data eddies: An adaptive web query processing engine for RDF data. In M. Arenas et al., editors, *The Semantic Web - ISWC 2015*, volume 9366 of *Lecture Notes in Computer Science*, pages 111–127. Springer International Publishing, 2015.

[3] M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. Anapsid: An adaptive query processing engine for SPARQL endpoints. In L. Aroyo et al., editors, *Proceedings of the 10th International Semantic Web Conference*. Springer Berlin Heidelberg 2011, pages 18–34.

[4] Z. Akar, T.G. Halaç, E.E. Ekinci, and O. Dikenelli. Querying the web of interlinked datasets using VoID descriptions. In C. Bizer et al., editors, *WWW2012 Workshop on Linked Data on the Web*. Lyon France 2012.

[5] M. Arenas, C. Gutierrez, and J. Pérez. Foundations of RDF databases. In S. Tessaris et al., editors, *Reasoning Web. Semantic Technologies for Information Systems*, volume 5689 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg 2009, pages 158–204.

[6] C. Başca and A. Bernstein. Querying a messy web of data with avalanche. *Web Semantics: Science, Services and Agents on the World Wide Web* volume 26, pages 1–28. Elsevier 2014.

[7] C. Basca and A. Bernstein. Avalanche: Putting the spirit of the web back into semantic web querying. *The 9th International Semantic Web Conference, Posters & Demo Session*. Bonn Germany 2010.

[8] H. Betz, F. Gropengießer, K. Hose, and K.-U. Sattler. Learning from the history of distributed query processing - a heretic view on linked data management. In *Proceedings of the 3rd International Workshop on Consuming Linked Data*. Boston, MA, USA 2012.

[9] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems* volume 5, pages 1–22. IGI Publishing 2009.

[10] Diego Calvanese, Martin Giese, Dag Hovland, and Martin Rezk. Ontology-based integration of cross-linked datasets. In M. Arenas et al., editors, *Proc. of the 14th Int. Semantic Web Conf. (ISWC 2015)*, volume 9366 of *Lecture Notes in Computer Science*, pages 199–216. Springer, 2015.

[11] X. Chen, H. Chen, N. Zhang, and S. Zhang. SparkRDF: Elastic discreted RDF graph processing engine with distributed memory. In M. Horridge et al., editors, *International Semantic Web Conference (Posters and Demos)*, volume 1272 of *CEUR Workshop Proceedings*, pages 261–264. CEUR-WS.org, 2014.

[12] H. Choi, J. Son, Y. Cho, M.K. Sung, and Y.D. Chung. SPIDER: a system for scalable, parallel / distributed evaluation of large-scale RDF data. In *Proceedings of the 18th ACM conference on Information and knowledge management*. New York, NY, USA, 2009, pages 2087–2088.

[13] O. Görlitz and S. Staab. SPLENDID: SPARQL Endpoint Federation Exploiting VoID Descriptions. In *Proceedings of the 2nd International Workshop on Consuming Linked Data*. Bonn Germany 2011.

[14] O. Görlitz, and S. Staab. Federated data management and query optimization for linked open data. *In* A. Vakali and L. C. Jain, eds, *New Directions in Web Data Management 1*, volume 331 of *Studies in Computational Intelligence*. Springer 2011, pages 109–137.

[15] Olaf Hartig and Giuseppe Pirro. A context-based semantics for SPARQL property paths over the web. In F. Gandon et al, editors, *The Semantic Web. Latest Advances and New Domains*, volume 9088 of *Lecture Notes in Computer Science*, pages 71–87. Springer International Publishing, 2015.

[16] O. Hartig. SPARQL for a web of linked data: Semantics and computability. In M. Sabou et al., editors, *Proceedings of the 9th International Conference on The Semantic Web: Research and Applications*. Springer-Verlag Berlin Heidelberg 2012, pages 8–23.

[17] O. Hartig. Zero-knowledge query planning for an iterator implementation of link traversal based query execution. In In G. Antinou et al., editors, *Proceedings of the 8th Extended Semantic Web Conference on The Semantic Web: Research and Applications*. Springer-Verlag Berlin Heidelberg 2011, pages 154–169.

[18] O. Hartig, C. Bizer, and J.C. Freytag. Executing SPARQL queries over the web of linked data. In A. Bernstein et al., editors, *Proceedings of the 8th International Semantic Web Conference*. Springer-Verlag, Berlin Heidelberg 2009, pages 293–309.

[19] T. Heath and C. Bizer. Linked data: Evolving the web into a global data space. *Synthesis Lectures on the Semantic Web: Theory and Technology*, 1(1):1–136, 2011.

[20] A. Hogan, M. Arenas, A. Mallea, and A. Polleres. Everything you always wanted to know about blank nodes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 27-28: 42 – 69. Elsevier 2014.

[21] K. Hose, R. Schenkel, M. Theobald, and G. Weikum. Database foundations for scalable RDF processing. In F. Wolfgang and L. Domenico, editors, *Proceedings of the 7th international conference on Reasoning web: semantic technologies for the web of data*. Springer-Verlag Berlin Heidelberg 2011, pages 202–249.

[22] J. Huang, D. J. Abadi, and K. Ren. Scalable SPARQL querying of large RDF graphs. In J. Blakeley et al., editors, *The Proceedings of the VLDB Endowment*, volume 4 number 11, 2011, pages 1123–1134.

[23] M.F. Husain, J. McGlothlin, M.M. Masud, L.R. Khan, and B. Thuraisingham. Heuristics-based query processing for large RDF graphs using cloud computing. In *IEEE Transactions on Knowledge and Data Engineering*, volume 23 issue 9, 2011, pages 1312–1327.

[24] G. Ladwig and T. Tran. SIHjoin: Querying remote and local linked data. In G. Antoniou et al., editors, *The Semantic Web: Research and Applications*, volume 6643 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg 2011, pages 139–153.

[25] C. Liu, J. Qu, G. Qi, H. Wang, and Y. Yu. Hadoopsparql: A hadoop-based engine for multiple SPARQL query answering. In E. Simperl et al., editors, *The Semantic Web: ESWC 2012 Satellite Events*, volume 7540 of *Lecture Notes in Computer Science*, pages 474–479. Springer Berlin Heidelberg, 2015.

[26] S. Lynden, I. Kojima, A. Matono, and Y. Tanimura. Adaptive integration of distributed Semantic Web data. In A. Madaan et al., editors, *8th International Workshop on Databases in Networked Information Systems*. Volume 7813 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg 2010, pages 174–193.

[27] D. C. Makinson. Completeness theorems, representation theorems: What's the difference? In Rønnow-Rasmussen et al., editor, *Hommage á Wlodek: Philosophical Papers dedicated to Wlodek Rabinowicz*. Electronic publication, 2007.

[28] G. Montoya, H. Skaf-Molli, P. Molli, and M.-E. Vidal. Federated sparql queries processing with replicated fragments. In M. Arenas et al., editors, *The Semantic Web - ISWC 2015*, volume 9366 of *Lecture Notes in Computer Science*, pages 36–51. Springer International Publishing, 2015.

[29] G. Montoya, M.-E. Vidal, and M. Acosta. A heuristic-based approach for planning federated SPARQL queries. In J. Sequeda et al., editors, *Proceedings of the 3rd International Workshop on Consuming Linked Data*, volume 905 of *CEUR Workshop Proceedings*. Boston, MA, USA, 2012.

[30] Y. Nenov, R. Piro, B. Motik, I. Horrocks, Z. Wu, and J. Banerjee. Rdfox: A highly-scalable rdf store. In M. Arenas et al., editors, *The Semantic Web - ISWC 2015*, volume 9367 of *Lecture Notes in Computer Science*, pages 3–20. Springer International Publishing, 2015.

[31] N. Papailiou, I. Konstantinou, D. Tsoumakos, and N. Koziris. H2rdf: adaptive query processing on RDF data in the cloud. In A. Mille et al., editors, *Proceedings of the 24th International World Wide Web Conference (Companion Volume)*. ACM 2012, pages 397–400.

[32] E. Prud'hommeaux, S. Harris, and A. Seaborne. SPARQL 1.1 Query Language. Technical report, W3C, 2013. URL `http://www.w3.org/TR/sparql11-query`.

[33] B. Quilitz and U. Leser. Querying distributed RDF data sources with SPARQL. In S. Bechhofer et al., editors, *Proceedings of the 5th European Semantic Web Conference*, Tenerife, Canary Islands, Spain. Volume 5021 of *Lecture Notes in Computer Science*. Springer 2008.

[34] A. G. Ryman, A. Le Hors, and S. Speicher. OSLC resource shape: A language for defining constraints on linked data. In *Proceedings of the Linked Data on the Web Workshop 2013*. Rio de Janeiro, Brazil 2013.

[35] A. Schatzle, M. Przyjaciel-Zablocki, T. Berberich, and G. Lausen. S2x: Graph-parallel querying of rdf with graphx. In *Proc. of 1st International Workshop on Big-Graphs Online Querying (Big-O(Q) 2015) at VLDB 2015*, 2015.

[36] S. Schenk, C. Saathoff, S. Staab, and A. Scherp. Semaplorer-interactive semantic exploration of data and media based on a federated cloud infrastructure. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(4):298 – 304, 2009. Elsevier 2008.

[37] M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran. FedBench: A benchmark suite for federated semantic data query processing. In L. Aroyo et al., editors, *Proceedings of the 10th International Semantic Web Conference*. Volume 7031 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg 2011, pages 585–600.

[38] A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. FedX: optimization techniques for federated query processing on linked data. In L. Aroyo et al., editors, *Proceedings of the 10th International Semantic Web Conference*. Springer Berlin Heidelberg 2011, pages pages 601–616.

[39] Audun Stolpe. A logical characterisation of SPARQL federation. *Semantic Web Journal*, 6(6), 2015.

[40] M. Vander Sande, R. Verborgh, J. Van Herwegen, E. Mannens, and R. Van de Walle. Opportunistic linked data querying through approximate membership metadata. In M. Arenas et al., editors, *The Semantic Web - ISWC 2015*, volume 9366 of *Lecture Notes in Computer Science*, pages 92–110. Springer International Publishing, 2015.

[41] R. Verborgh, O. Hartig, B. De Meester, G. Haesendonck, L. De Vocht, M. Vander Sande, R. Cyganiak, P. Colpaert, E. Mannens, and R. Van de Walle. Querying datasets on the web with high availability. In P. Mika et al., editors, *The Semantic Web — ISWC 2014*, volume 8796 of *Lecture Notes in Computer Science*, pages 180–196. Springer International Publishing, 2014.

[42] R. Verborgh, Miel Vander Sande, Pieter Colpaert, Sam Coppens, Erik Mannens, and Rik Van de Walle. Web-scale querying through Linked Data Fragments. In *Proceedings of the 7th Workshop on Linked Data on the Web*, April 2014.