Completeness Statements about RDF Data Sources and Their Use for Query Answering

Fariz Darari, Werner Nutt, Giuseppe Pirrò, and Simon Razniewski

Free University of Bozen-Bolzano, Bozen-Bolzano, Italy
fariz.darari@stud-inf.unibz.it, {nutt,pirro,razniewski}@inf.unibz.it

Abstract. With thousands of RDF data sources available on the Web covering disparate and possibly overlapping knowledge domains, the problem of providing high-level descriptions (in the form of metadata) of their content becomes crucial. In this paper we introduce a theoretical framework for describing data sources in terms of their completeness. We show how existing data sources can be described with completeness statements expressed in RDF. We then focus on the problem of the completeness of query answering over plain and RDFS data sources augmented with completeness statements. Finally, we present an extension of the completeness framework for federated data sources.

1 Introduction

The Resource Description Framework (RDF) [9] is the standard data model for the publishing and interlinking of data on the Web. It enables the making of *statements* about resources in the form of triples including a *subject*, a *predicate* and an *object*. Ontology languages such as RDF Schema (RDFS) and OWL provide the necessary underpinning for the creation of vocabularies to structure knowledge domains. RDF is now a reality; efforts like the Linked Open Data project [8] give a glimpse of the magnitude of RDF data today available online. The common path to access such huge amount of structured data is via SPARQL endpoints, that is, network locations that can be queried upon by using the SPARQL query language [5].

With thousands of RDF data sources covering possibly overlapping knowledge domains, the problem of providing high-level descriptions (in the form of metadata) of their content becomes crucial. Such descriptions will connect data publishers and consumers; publishers will advertise "what" is there inside a data source so that specialized applications can be created for data source discovering, cataloging, selection and so forth. Proposals like the `VoID` [1] vocabulary touched this aspect. With `VoID` it is possible to provide statistics about how many instances a particular *class* has, information about its SPARQL endpoint and links with other data sources, among the other things. However, `VoID` mainly focuses on providing *quantitative* information. We claim that toward comprehensive descriptions of data sources, *qualitative* information is crucial.

Related Work. Data quality is about the "fitness for use" of data and encompasses several dimensions such as accuracy, correctness and completeness.

H. Alani et al. (Eds.): ISWC 2013, Part I, LNCS 8218, pp. 66–83, 2013.

Fürber and Hepp [4] investigated data quality problems for RDF data originating from relational databases, while Wang et al. [19] focused on data cleansing. The problem of assessing completeness of Linked Data sources was discussed by Harth and Speiser [6]; here, completeness is defined in terms of *authoritativeness* of data sources, which is a purely syntactic property. Polleres et al. [16] defined a rule language where the need for completeness information emerges. Hartig et al. [7] discussed an approach to get more complete results of SPARQL queries over the Web of Linked Data. Their approach is based on traversing RDF links to discover relevant data during query execution. Still, the completeness of query answers cannot be guaranteed. In the relational databases world, completeness was first investigated by Motro [12] who provided a formalization of completeness of databases and queries. Halevy [11] studied the problem of how statements of completeness about a database related to query completeness. Recently, Razniewski and Nutt [17] provided a general solution to this problem, including a comprehensive study of the complexity of reasoning.

Indeed, the semantics of completeness is crucial also for RDF data sources distributed on the Web, where each data source is generally considered incomplete. To the best of our knowledge, the problem of formalizing the semantics of RDF data sources in terms of their completeness is open. Also from the more pragmatic point of view, there exist no comprehensive solutions enabling the characterization of data source in terms of completeness. As an example, with `VoID` it is not possible to express that, for instance, the data source IMDb is *complete for all movies directed by Tarantino*. Having the possibility to provide in a declarative and machine-readable way (in RDF), such kind of completeness statements paves the way toward a new generation of services for retrieving and consuming data. In this latter respect, the semantics of completeness statements interpreted by a reasoning engine can guarantee the completeness of query answering. We present a comprehensive application scenario in Section 2.

Contributions. This paper lays the foundation for the expression of completeness statements about RDF data sources. It can complement, with *qualitative* descriptions, existing proposals like `VoID` that mainly deal with *quantitative* descriptions. We develop a formalism and show its feasibility. The second goal of this paper is to show how completeness statements can be useful in practice. In this respect, we focus on the problem of query completeness. We believe that our research has both a theoretical and practical impact. On the theoretical side, we provide a formalization of completeness for RDF data sources and techniques to reason about the completeness of query answers in various settings, from plain RDF to federated data sources. From the practical side, completeness statements can be easily embedded in current descriptions of data sources and thus readily used. Finally, we want to point out that our completeness framework has been implemented in the `CoRNER` system, which is available for download[1].

Outline. In Section 2 we discuss a real world scenario and provide a high level overview of the completeness framework. Section 3 after providing some

[1] `http://rdfcorner.wordpress.com/`

background introduces a formalization of the completeness problem for RDF data sources. This section also describes how completeness statements can be represented in RDF. In Section 4 we discuss how completeness statements can be used in query answering when considering a single data source at a time. In Section 5 we challenge query completeness in federated data sources. Section 6 contains a discussion and Section 7 the conclusions.

2 Motivating Scenario

In this section we motivate the need of formalizing and expressing completeness statements in a machine-readable way. Moreover we show how completeness statement are useful for query answering. We start our discussion with a real data source available on the Web. Fig. 1 shows a screenshot taken from the IMDb website. The page is about the movie Reservoir Dogs; in particular it lists the cast and crew of the movie. For instance, it says that Tarantino was not only the director and writer of the movie but also the character Mr. Brown. As it can be noted, the data source includes a "completeness statement", which says that the page is *complete for all cast and crew members of the movie*. The availability of such statement increases the potential value of the data source. In particular, users who were looking for information about the cast of this movie and found this page can prefer it to other pages since, assuming the truth of the statement, all they need is here.

Fig. 1. A *completeness statement* in IMDb as of 7 May 2013. It says that the source is complete for the cast and crew of the movie *Reservoir Dogs*.

The problem with such kind of statements, expressed in natural language, is that they cannot be automatically processed, thus hindering their applicability, for instance, in query answering. Indeed, the interpretation of the statement "verified as complete" is left to the user. On the other hand, a reasoning and querying engine when requested to provide information about the cast and crew

members of Reservoir Dogs could have leveraged such statement and inform the user about the completeness of the results.

Other examples of Web data sources that already provide completeness statements are OpenStreetMap[2] and Wikipedia, which has, for instance, a complete list of works attributed to *Vermeer* and works by *Shakespeare* or a complete list of Olympic medalists in archery from 1900 to 2012. If such statements were exploited by machines, one would expect that there would be an incentive to publish them.

Machine-Readable Statements. In the RDF and Linked Data context with generally incomplete and possibly overlapping data sources and where *"anyone can say anything about any topic and publish it anywhere"* [9] having the possibility to express completeness statements becomes an essential aspect. The machine-readable nature of RDF enables to deal with the problems discussed in the example about IMDb; completeness statements can be represented in RDF. As an example, the high-level description of a data source like DBpedia could include, for instance, the fact that it is complete for all of Quentin Tarantino's movies. Fig. 2 shows how the data source DBpedia can be complemented with completeness statements expressed in our formalism. Here we give a high level presentation of the completeness framework; details on the theoretical framework supporting it are given in Section 3.

```
@prefix       c: <http://inf.unibz.it/ontologies/completeness#> .
@prefix     rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix    rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix  schema: <http://schema.org/> .
@prefix    spin: <http://spinrdf.org/sp#> .
@prefix     dbp: <http://dbpedia.org/resource/> .
@prefix    void: <http://rdfs.org/ns/void#> .
@prefix      dv: <http://dbpedia.org/void/> .

dv:dbpdataset rdf:type void:Dataset .

dv:dbpdataset rdfs:comment "This document provides completeness statements
about the dbpedia.org datasource" .

dv:dbpdataset c:hasComplStmt dv:st1.
dv:st1        c:hasPattern    [c:subject   [spin:varName "m"];
                                           c:predicate rdf:type;
                                           c:object    schema:Movie     ].
dv:st1        c:hasPattern    [c:subject   [spin:varName "m"];
                                           c:predicate schema:director;
                                           c:object    dbp:Tarantino].
dv:st1        rdfs:comment "This completeness statement indicates that
dbpedia.org is complete for all movies directed by Tarantino".
```

Fig. 2. An example of completeness statement about `dbpedia.org`

A simple statement can be thought of as a SPARQL Basic Graph Pattern (BGP). The BGP `(?m rdf:type schema:Movie).(?m schema:director dbp:Tarantino)`, for instance, expresses the fact that `dbpedia.org` is complete for all movies directed by Tarantino. In the figure, this information is

[2] `http://wiki.openstreetmap.org/wiki/Hall_of_Fame/Streets_complete`

represented by using an ad-hoc completeness vocabulary (see Section 3.2) with some properties taken from the SPIN[3] vocabulary.

Query Completeness. The availability of completeness statements about data sources is useful in different tasks, including data integration, data source discovery and query answering. In this paper we will focus on how to leverage completeness statements for query answering. The research question we address is how to assess whether available data sources with different degree of completeness can ensure the completeness of query answers. Consider the scenario depicted in Fig. 3 where the data sources DBpedia and LinkedMDB are described in terms of their completeness. The Web user Syd wants to pose the query Q to the SPARQL endpoints of these two data sources asking for *all movies directed by Tarantino in which Tarantino also starred*. By leveraging the completeness statements, the query engines at the two endpoints could tell Syd whether the answer to his query is complete or not. For instance, although DBpedia is complete for all of Tarantino's movies (see Fig. 2) nothing can be said about his participation as an actor in these movies (which is required in the query). Indeed, at the time of writing this paper, DBpedia is actually incomplete; this is because in the description of the movie Reservoir Dogs the fact is missing that Tarantino was the character Mr. Brown (and from Fig. 1 we know that this is the case). On the other hand, LinkedMDB, the RDF counterpart of IMDb, can provide a complete answer. Indeed, with our framework it is possible to express in RDF the completeness statement available in natural language in Fig. 1. This statement has then been used by the `CoRNER` reasoning engine, implementing our formal framework, to state the completeness of the query.

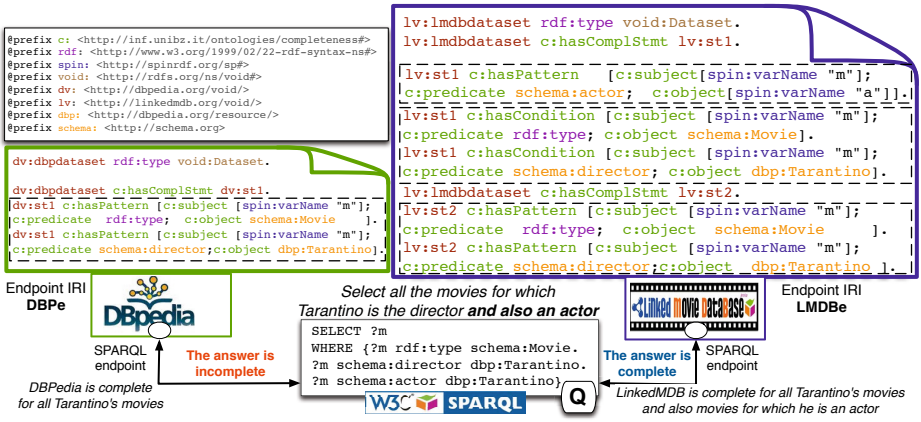

Fig. 3. Completeness statements and their usage for query answering

In this specific case, LinkedMDB can guarantee the completeness of the query answer because it contains all the actors in Tarantino's movies (represented by

[3] `http://spinrdf.org/sp.html#sp-variables`

the statement `lv:st1`) in addition to the Tarantino's movies themselves (represented by the statement `lv:st2`). Note that the statement `lv:st1` includes two parts: (i) the pattern, which is expressed via the BGP `(?m, schema:actor, ?a)` and (ii) the conditions, that is, the BGP `(?m, rdf:type, schema:Movie).(?m, schema:director, dbp:Tarantino)`. Indeed, a completeness statement allows one to say that a certain part (i.e., with respect to some conditions) of data is complete, or in other words, it can be used to state that a data source contains all triples in a pattern P_1 that satisfy a condition P_2. The detailed explanation and the semantics of completeness statements can be found in Section 3.

Application Scenarios. Completeness statements are particularly useful for data collections such as works of an artist, cities in countries, election results, census data and so forth. Completeness statements have wide applicability. *Source selection*: as an example for address verification, one needs a complete set of street names; for Hamburg, Dresden, and other cities in Germany, OpenStreetMap can be used because completeness is asserted. *Search Optimization*: a user wants to look for movies by Tarantino in 2008. By having completeness statements in IMDb about these movies, a search engine could stop after finding this specific source without the need to consult other sources.

3 Formal Framework

In the following, we remind the reader of RDF and SPARQL, formalize our framework and show how completeness information can be expressed in RDF.

RDF and SPARQL. We assume that there are three pairwise disjoint infinite sets I (*IRIs*), L (*literals*) and V (*variables*). We collectively refer to IRIs and literals as *RDF terms* or simply *terms*. A tuple $(s, p, o) \in I \times I \times (I \cup L)$ is called an *RDF triple* (or a *triple*), where s is the *subject*, p the *predicate* and o the *object* of the triple. An *RDF graph* or *data source* consists of a finite set of triples [9]. For simplicity, we omit namespaces for the abstract representation of RDF graphs.

The standard query language for RDF is SPARQL. The basic building blocks of a SPARQL query are *triple patterns,* which resemble RDF triples, except that in each position also variables are allowed. SPARQL queries include *basic graph patterns* (BGP), built using the `AND` operator, and more sophisticated operators, including `OPT`, `FILTER`, `UNION` and so forth. In this paper we consider the operators `AND` and `OPT`. Moreover, we also consider the result modifier `DISTINCT`. Evaluating a graph pattern P over an RDF graph G results in a set of mappings from the variables in P to terms, denoted as $[\![P]\!]_G$. Further information about SPARQL can be found in [14].

SPARQL queries come as `SELECT`, `ASK`, or `CONSTRUCT` queries. A `SELECT` query has the abstract form (W, P), where P is a graph pattern and W is a subset of the variables in P. A `SELECT` query $Q = (W, P)$ is evaluated over a graph G by restricting the mappings in $[\![P]\!]_G$ to the variables in W. The result is denoted as $[\![Q]\!]_G$. Syntactically, an `ASK` query is a special case of a `SELECT` query where

W is empty. For an `ASK` query Q, we write also $[\![Q]\!]_G = \texttt{true}$ if $[\![Q]\!]_G \neq \emptyset$, and $[\![Q]\!]_G = \texttt{false}$ otherwise. A `CONSTRUCT` query has the abstract form (P_1, P_2), where P_1 is a BGP and P_2 is a graph pattern. In this paper, we only use `CONSTRUCT` queries where also P_2 is a BGP. The result of evaluating $Q = (P_1, P_2)$ over G is the graph $[\![Q]\!]_G$, that is obtained by instantiating the pattern P_1 with all the mappings in $[\![P_2]\!]_G$.

Later on, we will distinguish between three classes of queries: (i) Basic queries, that is, queries (W, P) where P is a BGP and which return bags of mappings (as it is the default in SPARQL), (ii) `DISTINCT` queries, that is, queries $(W, P)^d$ where P is a BGP and which return sets of mappings, and (iii) `OPT` queries, that is, queries (W, P) without projection ($W = var(P)$) where P is a graph pattern with `OPT`.

3.1 Completeness Statements and Query Completeness

We are interested in formalizing when a query is complete over a potentially incomplete data source and in describing which parts of such a source are complete. When talking about the completeness of a source, one implicitly compares the information *available* in the source with what holds in the world and therefore should *ideally* be also present in the source. In this paper, we only consider sources that may miss information, but do not contain wrong information.

Definition 1 *(**Incomplete Data Source**). We identify data sources with RDF graphs. Then, adapting a notion introduced by Motro in [12], we define an incomplete data source as a pair $\mathcal{G} = (G^a, G^i)$ of two graphs, where $G^a \subseteq G^i$. We call G^a the* available *graph and G^i the* ideal *graph.*

Example 2 *(**Incomplete Data Source**).* Consider the DBpedia data source and suppose that the only movies directed by Tarantino are Reservoir Dogs, Pulp Fiction, and Kill Bill, and that Tarantino was starred exactly in the movies Desperado, Reservoir Dogs, and Pulp Fiction. For the sake of example, suppose also the fact that he was starred in Reservoir Dogs is missing in DBpedia[4]. Using Definition 1, we can formalize the incompleteness of the DBpedia data source $\mathcal{G}_{dbp}$ as:

$$
\begin{aligned}
G^a_{dbp} = \{&(\mathit{reservoirDogs}, \mathit{director}, \mathit{tarantino}), (\mathit{pulpFiction}, \mathit{director}, \mathit{tarantino}),\\
&(\mathit{killBill}, \mathit{director}, \mathit{tarantino}), (\mathit{desperado}, \mathit{actor}, \mathit{tarantino}),\\
&(\mathit{pulpFiction}, \mathit{actor}, \mathit{tarantino}), (\mathit{desperado}, \mathit{type}, \mathit{Movie}),\\
&(\mathit{reservoirDogs}, \mathit{type}, \mathit{Movie}), (\mathit{pulpFiction}, \mathit{type}, \mathit{Movie}), (\mathit{killBill}, \mathit{type}, \mathit{Movie})\}\\
G^i_{dbp} = G^a_{dbp} \cup \{&(\mathit{reservoirDogs}, \mathit{actor}, \mathit{tarantino})\}
\end{aligned}
$$

We now introduce *completeness statements*, which are used to denote the partial completeness of a data source, that is, they describe for which parts the ideal and available graph coincide.

[4] As it was the case on 7 May 2013.

Definition 3 ***(Completeness Statement).*** *A completeness statement* $Compl(P_1 \mid P_2)$ *includes:* P_1 *a non-empty BGP and* P_2 *a BGP. We call* P_1 *the* pattern *and* P_2 *the* condition *of the completeness statement.*

For example, we express that a source is complete for all pairs of triples that say "?m *is a movie and* ?m *is directed by Tarantino*" using the statement

$$C_{dir} = Compl((?m, type, Movie), (?m, director, tarantino) \mid \emptyset), \tag{1}$$

whose pattern matches all such pairs and whose condition is empty. To express that a source is complete for all triples about actors in movies directed by Tarantino, we use

$$C_{act} = Compl((?m, actor, ?a) \mid (?m, director, tarantino), (?m, type, Movie)), \tag{2}$$

whose pattern matches triples about actors and the condition restricts the actors to movies directed by Tarantino. The condition in C_{act} means that the data source does not necessarily contain triples of the form $(?m, director, tarantino)$ and $(?m, type, Movie)$. Moving the condition to the pattern imposes that the data source contains the triples.

We now define when a completeness statement is satisfied by an incomplete data source. To a statement $C = Compl(P_1 \mid P_2)$, we associate the `CONSTRUCT` query $Q_C = (P_1, P_1 \cup P_2)$. Note that, given a graph G, the query Q_C returns those instantiations of the pattern P_1 that are present in G together with an instantiation of the condition. For example, the query $Q_{C_{act}}$ returns all the acting information of Tarantino movies in G.

Definition 4 ***(Satisfaction of Completeness Statements).*** *For an incomplete data source* $\mathcal{G} = (G^a, G^i)$*, the statement* C *is satisfied by* $\mathcal{G}$*, written* $\mathcal{G} \models C$*, if* $[\![Q_C]\!]_{G^i} \subseteq G^a$ *holds.*

To see that the statement C_{dir} is satisfied by $\mathcal{G}_{dbp}$, observe that the query $Q_{C_{dir}}$ returns over G^i_{dbp} all triples with the predicate *actor* and all *type* triples for Tarantino movies, and that all these triples are also in G^a_{dbp}. However, C_{act} is *not* satisfied by $\mathcal{G}_{dbp}$, because $Q_{C_{act}}$ returns over G^i_{dbp} the triple $(reservoirDogs, actor, tarantino)$, which is not in G^a_{dbp}.

When querying a potentially incomplete data source, we would like to know whether at least the answer to our query is complete. For instance, when querying DBpedia for movies starring Tarantino, it would be interesting to know whether we really get all such movies, that is, whether our query is complete over DBpedia. We next formalize query completeness with respect to incomplete data sources.

Definition 5 ***(Query Completeness).*** *Let* Q *be a* `SELECT` *query. To express that* Q is complete*, we write* $Compl(Q)$*. An incomplete data source* $\mathcal{G} = (G^a, G^i)$ *satisfies the expression* $Compl(Q)$*, if* Q *returns the same result over* G^a *as it does over* G^i*, that is* $[\![Q]\!]_{G^a} = [\![Q]\!]_{G^i}$*. In this case we write* $\mathcal{G} \models Compl(Q)$*.*

Example 6 ***(Query Completeness).*** Consider the incomplete data source $\mathcal{G}_{dbp}$ and the two queries Q_{dir}, asking for all movies directed by Tarantino, and $Q_{dir+act}$, asking for all movies, both directed by and starring Tarantino:

$$Q_{dir} = (\{\, ?m \,\}, \{\, (?m, type, Movie), (?m, director, tarantino) \,\})$$
$$Q_{dir+act} = (\{\, ?m \,\}, \{\, (?m, type, Movie), (?m, director, tarantino), (?m, actor, tarantino) \,\}).$$

Then, it holds that Q_{dir} is complete over $\mathcal{G}_{dbp}$ while $Q_{dir+act}$ is not. Later on, we show how to deduce query completeness from completeness statements.

3.2 RDF Representation of Completeness Statements

Practically, completeness statements should be compliant with the existing ways of giving metadata about data sources, for instance, by enriching the `VoID` description [1]. Therefore, it is essential to express completeness statements in RDF itself. Suppose we want to express that LinkedMDB satisfies the statement:

$$C_{act} = Compl((?m, actor, ?a) \mid (?m, type, Movie), (?m, director, tarantino)).$$

Then, we need a vocabulary to say that this is a statement about LinkedMDB, which triple patterns make up its pattern, and which its condition. We also need the vocabulary to represent the constituents of the triple patterns, namely subject, predicate, and object of a pattern. Therefore, we introduce the property names whose meaning is intuitive:

`hasComplStmt`, `hasPattern`, `hasCondition`, `subject`, `predicate`, `object`

If the constituent of a triple pattern is a term (an IRI or a literal), then it can be specified directly in RDF. Since this is not possible for variables, we represent a variable by a resource that has a literal value for the property `varName`. Now, we can represent C_{act} in RDF as the resource `lv:st1` described in Figure 3.

More generally, consider a completeness statement $Compl(P_1 \mid P_2)$, where $P_1 = \{\, t_1, \ldots, t_n \,\}$ and $P_2 = \{\, t_{n+1}, \ldots, t_m \,\}$ and each t_i, $1 \leq i \leq m$, is a triple pattern. Then the statement is represented using a resource for the statement and a resource for each of the t_i that is linked to the statement resource by the property `hasPattern` or `hasCondition`, respectively. The constituents of each t_i are linked to t_i's resource in the same way via `subject`, `predicate`, and `object`. All resources can be either IRIs or blank nodes.

4 Completeness Reasoning over a Single Data Source

In this section, we show how completeness statements can be used to judge whether a query will return a complete answer or not. We first focus on completeness statements that hold on a *single* data source, while completeness statements in the federated setting are discussed in Section 5.

Problem Definition. Let $\mathbf{C}$ be a set of completeness statements and Q be a `SELECT` query. We say that $\mathbf{C}$ *entails the completeness of* Q, written $\mathbf{C} \models \mathit{Compl}(Q)$, if any incomplete data source that satisfies $\mathbf{C}$ also satisfies $\mathit{Compl}(Q)$.

Example 7. Consider C_{dir} from (1). Whenever an incomplete data source $\mathcal{G}$ satisfies C_{dir}, then G^a contains all triples about movies directed by Tarantino, which is exactly the information needed to answer query Q_{dir} from Example 6. Thus, $\{ C_{dir} \} \models \mathit{Compl}(Q_{dir})$. This may not be enough to completely answer $Q_{dir+act}$, thus $\{ C_{dir} \} \not\models \mathit{Compl}(Q_{dir+act})$. We will now see how this intuitive reasoning can be formalized.

4.1 Completeness Entailment for Basic Queries

To characterize completeness entailment, we use the fact that completeness statements have a correspondence in `CONSTRUCT` queries. For any set $\mathbf{C}$ of completeness statements we define the operator $T_{\mathbf{C}}$ that maps graphs to graphs:

$$T_{\mathbf{C}}(G) = \bigcup_{C \in \mathbf{C}} Q_C(G)$$

Notice that for any graph G, the pair $(T_{\mathbf{C}}(G), G)$ is an incomplete data source satisfying $\mathbf{C}$ and $T_{\mathbf{C}}(G)$ is the smallest set (wrt. set inclusion) for which this holds.

Example 8 ***(Completeness Entailment).*** Consider the set of completeness statements $\mathbf{C}_{dir,act} = \{ C_{dir}, C_{act} \}$ and the query $Q_{dir+act}$. Recall that the query has the form $Q_{dir+act} = (\{ ?m \}, P_{dir+act})$, where $P_{dir+act} = \{ (?m, \mathit{type}, \mathit{Movie}), (?m, \mathit{director}, \mathit{tarantino}), (?m, \mathit{actor}, \mathit{tarantino}) \}$. We want to check whether these statements entail the completeness of $Q_{dir+act}$, that is, whether $\mathbf{C}_{dir,act} \models \mathit{Compl}(Q_{dir+act})$ holds. Suppose that $\mathcal{G} = (G^a, G^i)$ satisfies $\mathbf{C}_{dir,act}$. Suppose also that $Q_{dir+act}$ returns a mapping $\mu = \{ ?m \mapsto m' \}$ over G^i for some term m'. Then G^i contains $\mu P_{dir+act}$, the instantiation by μ of the BGP of our query, consisting of the three triples $(m', \mathit{type}, \mathit{Movie})$, $(m', \mathit{director}, \mathit{tarantino})$, and $(m', \mathit{actor}, \mathit{tarantino})$.

The `CONSTRUCT` query $Q_{C_{dir}}$, corresponding to our first completeness statement, returns over $\mu P_{dir+act}$ the two triples $(m', \mathit{type}, \mathit{Movie})$ and $(m', \mathit{director}, \mathit{tarantino})$, while the `CONSTRUCT` query $Q_{C_{act}}$, corresponding to the second completeness statement, returns the triple $(m', \mathit{actor}, \mathit{tarantino})$. Thus, all triples in $\mu P_{dir+act}$ have been reconstructed by $T_{\mathbf{C}_{dir,act}}$ from $\mu P_{dir+act}$.

Now, we have $\mu P_{dir+act} = T_{\mathbf{C}_{dir,act}}(\mu P_{dir+act}) \subseteq T_{\mathbf{C}_{dir,act}}(G^i) \subseteq G^a$, where the last inclusion holds due to $\mathcal{G} \models \mathbf{C}_{dir,act}$. Therefore, our query $Q_{dir+act}$ returns the mapping μ also over G^a. Since μ and $\mathcal{G}$ were arbitrary, this shows that $\mathbf{C}_{dir,act} \models \mathit{Compl}(Q_{dir+act})$ holds.

In summary, in Example 8 we have reasoned about a set of completeness statements $\mathbf{C}$ and a query $Q = (W, P)$. We have considered a generic mapping μ, defined on the variables of P, and applied it to P, thus obtaining a graph μP.

Then we have verified that $\mu P = T_{\mathbf{C}}(\mu P)$. From this, we could conclude that for every incomplete data source $\mathcal{G} = (G^a, G^i)$ we have that $[\![Q]\!]_{G^a} = [\![Q]\!]_{G^i}$. Next, we make this approach formal.

Definition 9 ***(Prototypical Graph).*** *Let (W, P) be a query. The* freeze mapping *$\tilde{id}$ is defined as mapping each variable v in P to a new IRI $\tilde{v}$. Instantiating the graph pattern P with $\tilde{id}$ yields the RDF graph $\tilde{P} := \tilde{id}\, P$, which we call the prototypical graph of P.*

Now we can generalize the reasoning from above to a generic completeness check.

Theorem 10 (Completeness of Basic Queries). *Let $\boldsymbol{C}$ be a set of completeness statements and let $Q = (W, P)$ be a basic query. Then*

$$\boldsymbol{C} \models \mathit{Compl}(Q) \qquad \text{if and only if} \qquad \tilde{P} = T_{\boldsymbol{C}}(\tilde{P}).$$

Proof. (Sketch) "⇒" If $\tilde{P} \neq T_{\mathbf{C}}(\tilde{P})$, then the pair $(T_{\mathbf{C}}(\tilde{P}), \tilde{P})$ is a counterexample for the entailment. It satisfies $\mathbf{C}$, but does not satisfy $\mathit{Compl}(Q)$ because the freeze mapping $\tilde{id}$ cannot be retrieved by P over the available graph $T_{\mathbf{C}}(\tilde{P})$.

"⇐" If all triples of the pattern $\tilde{P}$ are preserved by $T_{\mathbf{C}}$, then this serves as a proof that in any incomplete data source all triples that are used to compute a mapping in the ideal graph are also present in the available graph.

Queries with `DISTINCT`. Basic queries return bags of answers (i.e., they may contain duplicates), while `DISTINCT` eliminates duplicates. For a query Q involving `DISTINCT`, the difference to the characterization in Theorem 10 is that instead of retrieving the full pattern $\tilde{P}$ after applying $T_{\mathbf{C}}$, we only check whether sufficient parts of $\tilde{P}$ are preserved that still allow to retrieve the freeze mapping on the distinguished variables of Q.

4.2 Completeness of Queries with the `OPT` Operator

One interesting feature of SPARQL is the `OPT` ("optional") operator. With `OPT` one can specify that parts of a query are only evaluated if an evaluation is possible, similarly to an outer join in SQL. For example, when querying for movies, one can also ask for the prizes they won, if any. The `OPT` operator is used substantially in practice [15]. Intuitively, the mappings for a pattern (P_1 `OPT` P_2) are computed as the union of all the bindings of P_1 together with the bindings for P_2 that are valid extensions, and including those bindings of P_1 that have no binding for P_2 that is a valid extension. For a formal definition of the semantics of queries with the `OPT` operator, see [10]. Completeness entailment for queries with `OPT` differs from that of queries without.

Example 11 ***(Completeness with `OPT`).*** Consider the following query with `OPT` $Q_{maw} = ((?m, \mathit{type}, \mathit{Movie})$ `OPT` $(?m, \mathit{award}, ?aw))$, asking for all movies and if available, also their awards. Consider also $C_{aw} = \mathit{Compl}((?m, \mathit{type}, \mathit{Movie}), (?m, \mathit{award}, ?aw) \mid \emptyset)$, the completeness statement

that expresses that all movies that have an award are complete and all awards of movies are complete. If the query Q_{maw} used `AND` instead of `OPT`, then its completeness could be entailed by C_{aw}. However with `OPT` in Q_{maw}, more completeness is required: Also those movies have to be complete that do not have an award. Thus, C_{aw} alone does not entail the completeness of Q_{maw}.

If one uses `OPT` without restrictions, unintuitive queries may result. Pérez et al. have introduced the class of so-called *well-designed graph patterns* that avoid anomalies that may otherwise occur [14]. Formally, a graph pattern P is well-designed if for every subpattern $P' = (P_1 \texttt{ OPT } P_2)$ of P and for every variable $?X$ occurring in P, the following condition holds: if $?X$ occurs both inside P_2 and outside P', then it also occurs in P_1. We restrict ourselves in the following to `OPT` queries with well-designed patterns, which we call well-designed queries. Graph patterns with `OPT` have a hierarchical structure that can be made explicit by so-called pattern trees. A *pattern tree* $\mathcal{T}$ is a pair $(T, \mathcal{P})$, where (i) $T = (N, E, r)$ is a tree with node set N, edge set E, and root $r \in N$, and (ii) $\mathcal{P}$ is a labeling function that associates to each node $n \in N$ a BGP $\mathcal{P}(n)$. We construct for each pattern P a corresponding pattern tree $\mathcal{T}$. Any `OPT`-pattern can be translated into a pattern tree and vice versa [10]. As an example, consider a pattern $((P_1 \texttt{ OPT } P_2) \texttt{ OPT } (P_3 \texttt{ OPT } P_4))$, where P_1 to P_4 are BGPs. Its corresponding pattern tree would have a root node labeled with P_1, two child nodes labeled with P_2 and P_3, respectively, and the P_3 node would have another child labeled with P_4.

Patterns and pattern trees can contain redundant triples. Letelier et al. [10] have shown that for every pattern tree $\mathcal{T}$ one can construct in polynomial time an equivalent well-designed pattern tree $\mathcal{T}^{NR}$ without redundant triples, which is called the NR-normal form of $\mathcal{T}$. For every node n in $\mathcal{T}$ we define the branch pattern P_n of n as the union of the labels of all nodes on the path from n to the root of $\mathcal{T}$. Then the *branch query* Q_n of n has the form (W_n, P_n), where $W_n = var(P_n)$.

Theorem 12 (Completeness of `OPT`-Queries). *Let $\boldsymbol{C}$ be a set of completeness statements. Let $Q = (W, P)$ be a well-designed `OPT`-query and $\mathcal{T}$ be an equivalent pattern tree in NR-normal form. Then*

$$\boldsymbol{C} \models Compl(Q) \quad \textit{iff} \quad \boldsymbol{C} \models Compl(Q_n) \quad \textit{for all branch queries } Q_n \textit{ of } \mathcal{T}.$$

Technically, this theorem allows to reduce completeness checking for an `OPT` query to linearly many completeness checks for basic queries.

4.3 Completeness Entailment under RDFS Semantics

RDFS (RDF Schema) is a simple ontology language that is widely used for RDF data [3]. RDFS information can allow additional inference about data and needs to be taken into account during completeness entailment.

Example 13 ***(RDF vs. RDFS).*** Consider the query $Q_{film} = (\{ ?m \}, \{ (?m, type, film) \})$, asking for all films, and the completeness statement

$C_{movie} = Compl((?m, type, movie) \mid \emptyset)$ saying that we are complete for all movies. A priori, we cannot conclude that C_{movie} entails the completeness of Q_{film}, because we do not know about the relationship between films and movies. When considering the RDFS statements $(film, subclass, movie)$ and $(movie, subclass, film)$ saying that all movies and films are equivalent, we can conclude that $\{C_{movie}\} \models Compl(Q_{film})$.

In the following, we rely on ρDF, which formalizes the core of RDFS [13]. The ρDF vocabulary contains the terms *subproperty*, *subclass*, *domain*, *range* and *type*. A *schema graph* S is a set of triples built using any of the ρDF terms, except *type*, as predicates.

We assume that schema information is not lost in incomplete data sources. Hence, for incomplete data sources it is possible to extract their ρDF schema into a separate graph. The *closure of a graph* G, that is, $cl_S(G)$ wrt. a schema S is the set of all triples that are entailed. The computation of this closure can be reduced to the computation of the closure of a single graph that contains both schema and non-schema triples as $cl_S(G) = cl(S \cup G)$. We now say that a set $\mathbf{C}$ of completeness statements *entails* the completeness of a query Q wrt. a ρDF schema graph S, if for all incomplete data sources (G^a, G^i) it holds that if $(cl_S(G^a), cl_S(G^i))$ satisfies $\mathbf{C}$ then it also satisfies $Compl(Q)$.

Therefore, the main difference to the previous entailment procedures is that the closure is computed to obtain entailed triples before and after the completeness operator $T_{\mathbf{C}}$ is applied. For a set of completeness statements $\mathbf{C}$ and a schema graph S, let $T^S_{\mathbf{C}}$ denote the function composition $cl_S \circ T_{\mathbf{C}} \circ cl_S$. Then the following holds.

Theorem 14 (Completeness under RDFS). *Let $\boldsymbol{C}$ be a set of completeness statements, $Q = (W, P)$ a basic query, and S a schema graph. Then*

$$\boldsymbol{C} \models_S Compl(Q) \qquad \text{if and only if} \qquad \tilde{P} \subseteq T^S_{\mathbf{C}}(\tilde{P}).$$

5 Completeness Reasoning over Federated Data Sources

Data on the Web is intrinsically distributed. Hence, the single-source query mechanism provided by SPARQL has been extended to deal with multiple data sources. In particular, the recent SPARQL 1.1 specification introduces the notion of query *federation* [18]. A federated query is a SPARQL query that is evaluated across several data sources, the SPARQL endpoints of which can be specified in the query.

So far, we have studied the problem of querying a *single* data source augmented with completeness statements. The federated scenario calls for an extension of the completeness framework discussed in Section 4. Indeed, the completeness statements available about each data source involved in the evaluation of a federated query must be considered to check the completeness of the federated query. This section discusses this aspect and presents an approach to

check whether the completeness of a non-federated query (i.e., a query without `SERVICE` operators) can be ensured with respect to the completeness statements on each data source. We also study the problem of rewriting a non-federated query into a federated version in the case in which the query is complete.

Federated SPARQL Queries. Before introducing the extension of the completeness framework, we formalize the notion of federated SPARQL queries. A federated query is a SPARQL query executed over a *federated graph*. Formally speaking, a federated graph is a family of RDF graphs $\bar{G} = (G_j)_{j \in J}$ where J is a set of IRIs. A federated SPARQL query (as for the case of a non-federated query) can be a `SELECT` or an `ASK` query [2]. In what follows, we focus on the conjunctive fragment (i.e., the `AND` fragment) of SPARQL with the inclusion of the `SERVICE` operator. Non-federated SPARQL queries are evaluated over graphs. In the federated scenario, queries are evaluated over a pair $(i, \bar{G})$, where the first component is an IRI associated to the initial SPARQL endpoint, and the second component is a federated graph. The semantics of graph patterns with `AND` and `SERVICE` operators is defined as follows:

$$\begin{aligned} [\![t]\!]_{(i,\bar{G})} &= [\![t]\!]_{G_i} \\ [\![P_1 \texttt{ AND } P_2]\!]_{(i,\bar{G})} &= [\![P_1]\!]_{(i,\bar{G})} \bowtie [\![P_2]\!]_{(i,\bar{G})} \\ [\![(\texttt{SERVICE } j \ P)]\!]_{(i,\bar{G})} &= [\![P]\!]_{(j,\bar{G})} \end{aligned}$$

where t ranges over all triple patterns and P, P_1, P_2 range over all graph patterns with `AND` and `SERVICE` operators. We denote federated queries as $\bar{Q}$.

5.1 Federated Completeness Reasoning Framework

We now extend our completeness reasoning framework to the federated setting. We assume from now on that the set of IRIs J is fixed and all indices are drawn from J.

Definition 15 (Incomplete Federated Data Source). *An* incomplete federated data source *(or incomplete FDS, for short) is a pair $\bar{\mathcal{G}} = (\bar{G}^a, G^i)$, consisting of an* available federated graph $\bar{G}^a = (G^a_j)_{j \in J}$ *and an* ideal graph G^i*, such that $G^a_j \subseteq G^i$ for all $j \in J$.*

This captures the intuition that the ideal graph represents all the facts that hold in the world, while each source contains a part of those facts. Note that the graphs of the sources may overlap, as is the case on the Web. Next, we adapt completeness statements so that they talk about a specific source.

Definition 16 (Indexed Completeness Statements). *An* indexed completeness statement *is a pair (C, k) where C is a completeness statement and $k \in J$ is an IRI. An indexed completeness statement is satisfied by an incomplete FDS if it is satisfied by the incomplete data source corresponding to the index, that is,*

$$((G^a_j)_{j \in J}, G^i) \models_{fed} (C, k) \quad \textit{iff} \quad (G^a_k, G^i) \models C.$$

This definition is naturally extended to sets $\bar{\boldsymbol{C}}$ of indexed completeness statements.

We associate to each federated query, federated graph, incomplete FDSs, and set of indexed completeness statements a non-federated version, the flattening.

Definition 17 (Flattening). *The* flattening $\bar{Q}^{fl}$ *of a federated query* $\bar{Q}$ *is obtained from* $\bar{Q}$ *by replacing recursively each occurrence of a service call* (`SERVICE` j P) *with the pattern* P. *The* flattening $\bar{G}^{fl}$ *of a federated graph* $\bar{G} = (G_j)_{j \in J}$ *is the union of the individual graphs, that is,* $\bar{G}^{fl} = \bigcup_{j \in J} G_j$. *The* flattening $\bar{\mathcal{G}}^{fl}$ *of an incomplete FDS* $\bar{\mathcal{G}} = (\bar{G}^a, G^i)$ *is the incomplete data source* $\bar{\mathcal{G}}^{fl} = ((\bar{G}^a)^{fl}, G^i)$ *whose available graph is the flattening of the available federated graph of* $\bar{\mathcal{G}}$. *The* flattening $\bar{\boldsymbol{C}}^{fl}$ *of a set* $\bar{\boldsymbol{C}}$ *of indexed completeness statements is the set* $\bar{\boldsymbol{C}}^{fl} = \{\, C \mid (C, k) \in \bar{\boldsymbol{C}} \}$, *where we ignore the indices.*

Note that the notion of federated entailment is different from the entailment between a set of completeness statements and a query defined in Section 4 in the sense that we now have to deal with indexed completeness statements.

Definition 18 (Federated Completeness and Entailment). *A federated query* $\bar{Q}$ *is* complete *over an incomplete FDS* $\bar{\mathcal{G}} = (\bar{G}^a, G^i)$, *written* $\bar{\mathcal{G}} \models_{fed} Compl(\bar{Q})$, *if* $[\![\bar{Q}]\!]_{(j_0, \bar{G}^a)} = [\![\bar{Q}^{fl}]\!]_{G^i}$ *for any IRI* $j_0 \in J$, *that is, the evaluation of* $\bar{Q}$ *over the available federated graph returns the same result as evaluating the flattening of* $\bar{Q}$ *over the ideal graph. If* $\bar{\boldsymbol{C}}$ *is an indexed set of completeness statements, then* $\bar{\mathbf{C}}$ entails $Compl(\bar{Q})$, *written* $\bar{\boldsymbol{C}} \models_{fed} Compl(\bar{Q})$, *if* $\bar{\mathcal{G}} \models_{fed} \bar{\boldsymbol{C}}$ *implies* $\bar{\mathcal{G}} \models_{fed} Compl(\bar{Q})$ *for all incomplete FDSs* $\bar{\mathcal{G}}$.

If Q is a basic query, then we say that Q is complete over $\bar{\mathcal{G}}$ if Q is complete over the flattening of $\bar{\mathcal{G}}$, that is, $\bar{\mathcal{G}} \models_{fed} Compl(Q)$ iff $\bar{\mathcal{G}}^{fl} \models Compl(Q)$. This means that Q is complete if evaluated over the *union* of all sources in the federation.

Proposition 19 (Completeness of Basic Queries). *Let* $\bar{\boldsymbol{C}}$ *be a set of indexed completeness statement and* Q *be a basic query. Then*

$$\bar{\boldsymbol{C}} \models_{fed} Compl(Q) \quad \textit{iff} \quad \bar{\boldsymbol{C}}^{fl} \models Compl(Q)$$

This means that we can check the completeness of a basic query with the criterion in Theorem 10 in Section 4.1. A federated query $\bar{Q}$ is a *federated version* of a basic query Q if $\bar{Q}^{fl} = Q$. In other words, by dropping the service calls from $\bar{Q}$ we obtain Q.

Theorem 20. ***(Smart Rewriting)***. *Let* $\bar{\boldsymbol{C}}$ *be a set of indexed completeness statement and* Q *be a basic query such that* $\bar{\boldsymbol{C}} \models_{fed} Compl(Q)$. *Then:*

1. *One can compute a federated version* $\bar{Q}$ *of* Q *such that* $\bar{\boldsymbol{C}} \models_{fed} Compl(\bar{Q})$.
2. *Moreover, whenever* $(\bar{G}^a, G^i) \models_{fed} \bar{\boldsymbol{C}}$, *then*

$$[\![Q]\!]_{\bigcup_{j \in J} G_j^a} \;=\; [\![\bar{Q}]\!]_{(j_0, \bar{G}^a)} \qquad \textit{for any } j_0 \in J.$$

To retrieve all answers for an arbitrary query, we have to evaluate each triple pattern over the union of all sources. For a complete query, the federated version evaluates each triple pattern only over a single source. Therefore, the evaluation of the federated version is in general much more efficient.

Example 21 ***(Federated Data Sources).*** Consider the two data sources shown in Fig. 3 plus an additional data source named FB (= Facebook) with the completeness statement $C_{fb} = Compl(\{(?m, likes, ?l)\} \mid \{(?m, type, Movie), (?m, director, tarantino)\})$ and the query: $Q_{fb} = (\{?m, ?l\}, \{(?m, type, Movie), (?m, director, tarantino), (?m, likes, ?l)\})$ that asks for the number of *likes* of Tarantino's movies.

In order to answer this query efficiently over the three data sources, whose endpoints are reachable at the IRIs `DBPe`, `LMDBe` and `FBe`, we compute a federated version $\bar{Q}_{fb}$. The completeness statements in Fig. 3 plus C_{fb} entail wrt. "$\models_{fed}$" the completeness of the query Q_{fb} (see Definition 18). By Theorem 20 we can compute a complete federated version $\bar{Q}_{fb}$, which in this case is $\bar{Q}_{fb} = (\{?m, ?l\}, \{$ `(SERVICE LMDBe` $\{(?m, type, Movie), (?m, director, tarantino)\}$`)}) AND (SERVICE FBe` $\{(?m, likes, ?l)\}$`)`, whose answer is complete.

6 Discussion

We now discuss some aspects underlying the completeness framework.

Availability of Completeness Statements. At the core of the proposed framework lies the availability of completeness statements. We have discussed in Section 2 how existing data sources like IMDb already incorporate such statements (Figure 1) and how they can be made machine-readable with our framework. The availability of completeness statements rests on the assumption that a domain "expert" has the necessary background knowledge to provide such statements.

We believe that it is in the interest of data providers to annotate their data sources with completeness statements in order to increase their value. Indeed, users can be more inclined to prefer data sources including "completeness marks" to other data sources. Moreover, in the era of crowdsourcing the availability of independent "ratings" from users regarding the completeness of data can also contribute (like in Wikipedia and OpenStreetMap), in a bottom up manner, to the description of the completeness of data sources. For instance, when looking up information about Stanley Kubrick in DBpedia, as a by-product users can provide feedback as to whether all of Kubrick's movies are present. One can also imagine approaches based on gamification.

Maintenance. If edits of a source are logged, log items could be automatically translated into updates of statements. For *non-authoritative* sources, temporal guards can be used; e.g., instead of saying "complete for all movies by Tarantino", one would say "complete for movies by Tarantino in 2010".

Complexity. All completeness checks presented in this paper are NP-complete. The hardness holds because classical conjunctive query containment can be encoded into completeness checking [17]; the NP upper bound follows because all completeness checks require conjunctive query evaluation at their core. In practice, we expect these checks to be fast, since queries and completeness statements are likely to be small. After all, this is the same complexity as the one of query evaluation and query optimization of basic queries, as implemented in practical database management systems.

Vocabulary Heterogeneity. In practice, a query may use a vocabulary different from that of some data sources. In this work, we assume the presence of a global schema. Indeed, one could use the `schema.org` vocabulary for queries, since it has already been mapped to other vocabularies (e.g., DBpedia).

The `CoRNER` Implementation. To show the feasibility of our proposal, we developed the `CoRNER` system. It implements the completeness entailment procedure for basic and `DISTINCT` queries with ρDF. The system is implemented in Java and uses the Apache Jena library. It is downloadable at `http://rdfcorner.wordpress.com`.

7 Concluding Remarks and Future Work

The availability of distributed and potentially overlapping RDF data sources calls for mechanisms to provide qualitative characterizations of their content. In this respect, we have identified *completeness* as one important dimension. The motivation underlying this work stems from the fact that although completeness information is present in some available data sources (e.g., IMDb discussed in Section 2) it is neither formally represented nor automatically processed. We have introduced a formal framework for the declarative specification of completeness statements about RDF data sources and underlined how the framework can complement existing initiatives like `VoID`. Then, we studied "how" completeness statements can be used in the problem of completeness of query answering. In this respect we considered queries over single and federated data sources and showed how to assess query completeness. We believe that our research can be the starting point of further investigation of the problem of completeness of information on the Web. Considering other application scenarios of completeness statements like data source integration and selection is in our research agenda.

References

1. Alexander, K., Cyganiak, R., Hausenblas, M., Zhao, J.: Describing linked datasets with the VoID vocabulary. Technical report, W3C (2011)
2. Arenas, M., Gutierrez, C., Pérez, J.: On the semantics of SPARQL. In: Semantic Web Information Management, pp. 281–307. Springer, Heidelberg (2010)
3. Brickley, D., Guha, R.V.: RDF vocabulary description language 1.0: RDF Schema. Technical report, W3C (2004)

4. Fürber, C., Hepp, M.: Using SPARQL and SPIN for data quality management on the Semantic Web. In: Abramowicz, W., Tolksdorf, R. (eds.) BIS 2010. LNBIP, vol. 47, pp. 35–46. Springer, Heidelberg (2010)
5. Harris, S., Seaborne, A.: SPARQL 1.1 query language. Technical report, W3C (2013)
6. Harth, A., Speiser, S.: On completeness classes for query evaluation on linked data. In: AAAI (2012)
7. Hartig, O., Bizer, C., Freytag, J.-C.: Executing SPARQL queries over the web of linked data. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 293–309. Springer, Heidelberg (2009)
8. Heath, T., Bizer, C.: Linked Data: Evolving the Web into a Global Data Space. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers (2011)
9. Klyne, G., Carroll, J.J.: Resource Description Framework (RDF): Concepts and abstract syntax. Technical report, W3C (2004)
10. Letelier, A., Pérez, J., Pichler, R., Skritek, S.: Static analysis and optimization of semantic web queries. In: PODS, pp. 89–100 (2012)
11. Levy, A.Y.: Obtaining complete answers from incomplete databases. In: Proc. VLDB, pp. 402–412 (1996)
12. Motro, A.: Integrity = Validity + Completeness. ACM TODS 14(4), 480–502 (1989)
13. Muñoz, S., Pérez, J., Gutierrez, C.: Simple and efficient minimal RDFS. J. Web Sem. 7(3), 220–234 (2009)
14. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. ACM TODS 34(3), 16 (2009)
15. Picalausa, F., Vansummeren, S.: What are real SPARQL queries like? In: SWIM (2011)
16. Polleres, A., Feier, C., Harth, A.: Rules with contextually scoped negation. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 332–347. Springer, Heidelberg (2006)
17. Razniewski, S., Nutt, W.: Completeness of queries over incomplete databases. PVLDB 4(11), 749–760 (2011)
18. Seaborne, A., Polleres, A., Feigenbaum, L., Williams, G.T.: SPARQL 1.1 federated query. Technical report, W3C (2013)
19. Hamilton, H.J., Wang, X., Bither, Y.: An ontology-based approach to data cleaning. Department of Computer Science, University of Regina (2005)