

Completeness Management for RDF Data Sources

FARIZ DARARI, Faculty of Computer Science, Universitas Indonesia

WERNER NUTT, KRDB, Free University of Bozen-Bolzano

GIUSEPPE PIRRÒ, ICAR-CNR

SIMON RAZNIEWSKI, Max Planck Institute for Informatics

The Semantic Web is commonly interpreted under the open-world assumption, meaning that information available (e.g., in a data source) captures only a subset of the reality. Therefore, there is no certainty about whether the available information provides a *complete* representation of the reality. The broad aim of this article is to contribute a formal study of how to describe the completeness of parts of the Semantic Web stored in RDF data sources. We introduce a theoretical framework allowing augmentation of RDF data sources with statements, also expressed in RDF, about their completeness. One immediate benefit of this framework is that now query answers can be complemented with information about their completeness. We study the impact of completeness statements on the complexity of query answering by considering different fragments of the SPARQL language, including the RDFS entailment regime, and the federated scenario. We implement an efficient method for reasoning about query completeness and provide an experimental evaluation in the presence of large sets of completeness statements.

CCS Concepts: • **Information systems** → **Incomplete data; Query languages; Resource Description Framework (RDF)**;

Additional Key Words and Phrases: RDF, SPARQL, query answering, data completeness, metadata

ACM Reference format:

Fariz Darari, Werner Nutt, Giuseppe Pirrò, and Simon Razniewski. 2018. Completeness Management for RDF Data Sources. *ACM Trans. Web* 12, 3, Article 18 (July 2018), 53 pages.

<https://doi.org/10.1145/3196248>

1 INTRODUCTION

The increasing amount of structured data made available on the Web is laying the foundation of a global-scale knowledge base. Projects such as Linked Open Data (LOD) (Heath and Bizer 2011), by inheriting some basic design principles of the Web (e.g., simplicity, decentralization), aim at making huge volumes of data available by the Resource Description Framework (RDF) standard data format (Klyne and Carroll 2004). RDF enables one to make *statements* about resources in the form of triples, consisting of a *subject*, *predicate*, and *object*. Ontology languages, such as RDF

This work was partially supported by the project MAGIC (2/40.3), funded by the Province of Bozen-Bolzano, and the project TaDaQua (1644), funded by the Free University of Bozen-Bolzano.

Authors' addresses: F. Darari, Faculty of Computer Science, Universitas Indonesia, Kampus UI Depok, Indonesia 16424; email: fariz@cs.ui.ac.id; W. Nutt, Faculty of Computer Science, Free University of Bozen-Bolzano, Dominikanerplatz 3 - piazza Domenicani, 3, Italy - 39100, Bozen-Bolzano; email: nutt@inf.unibz.it; G. Pirrò, Via P. Bucci, cubo 7/11C, Rende (CS) - 87036; email: pirro@icar.cnr.it; S. Razniewski, Max Planck Institute for Informatics, Saarland Informatics Campus, Building E1 4, Saarbrücken, Germany - 66123; email: srazniew@mpi-inf.mpg.de.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2018 ACM 1559-1131/2018/07-ART18 \$15.00

<https://doi.org/10.1145/3196248>

Schema (RDFS) (Brickley and Guha 2004) and OWL (Hitzler et al. 2012), provide the necessary underpinning for the creation of vocabularies to structure knowledge domains. The common path to access such a huge amount of structured data is via SPARQL endpoints, namely, network locations that can be queried by using the SPARQL query language (Harris and Seaborne 2013).

With a large number of RDF data sources covering possibly overlapping knowledge domains, it is natural to observe a wide range of data source quality; some data sources are manually curated while others result from crowdsourcing efforts or automatic extraction techniques (Bizer et al. 2009; Hoffart et al. 2011). In this setting, 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 such as the VoID vocabulary (Alexander et al. 2011) address this aspect. With VoID it is possible, among other things, to provide information about how many instances a particular class has, the SPARQL endpoint of a source, and links to other data sources. However, VoID mainly focuses on providing *quantitative* information. We claim that in attempting to provide comprehensive descriptions of data sources, *qualitative* information is crucial; hence, *the overall aim of this article is to study a specific aspect of data quality for RDF data sources, that is, completeness.*

In previous work, Fürber and Hepp (2010) investigated data quality problems for RDF data originating from relational databases. Wang et al. (2005) focused on data cleansing, while Stoilos et al. (2010) concentrated on incompleteness of reasoning tasks. The problem of assessing the completeness of Linked Data sources was discussed by Harth and Speiser (2012); here, completeness is defined in terms of authoritativeness of data sources, which is a purely syntactic property. Hartig et al. (2009), Hartig (2015), Hartig and Pirrò (2017), Fionda et al. (2015a), and Fionda et al. (2015b) discussed approaches to retrieve more complete results of SPARQL queries over the Web of Linked Data. Their approaches are based on traversing RDF links to discover more relevant data during query execution; still, the completeness of query answers cannot be guaranteed. In the relational database world, completeness was first investigated by Motro (1989), who provided a formalization of completeness of databases and queries. Levy (1996) studied how statements about the completeness of a database relate to query completeness. Razniewski and Nutt (2011) provided a general solution to this problem for the relational setting, including a comprehensive study of the complexity of completeness reasoning. To the best of our knowledge, neither the problem of formalizing the semantics of RDF data sources in terms of their completeness nor the problem of assessing query completeness in centralized and federated scenarios has been addressed before.

Information about completeness is beneficial for RDF data sources, where data is generally assumed to be incomplete. As an example, it would be possible to complement the answer to a query (e.g., retrieve movies directed by Quentin Tarantino) with information about the completeness of the answer (e.g., the movies in the answer are *all* movies directed by Tarantino). However, so far, there is no approach to characterizing data sources in terms of their completeness that is both conceptually well founded and practically applicable. For instance, with the widely used metadata format VoID, it is not possible to express that an RDF data source of the movie website IMDb¹ is *complete for all movies directed by Tarantino*. The possibility of providing in a declarative and machine-readable way this kind of completeness statement paves the way for a new generation of services for consuming data. In this respect, the semantics of completeness statements interpreted by a reasoning engine can guarantee the completeness of query answering. We now give an overview of our approach motivated by real-world scenarios.

¹<http://www.imdb.com/>.



Completeness statement about the IMDB data source

Quentin Tarantino was the character Mr. Brown

Fig. 1. A completeness statement (in natural language) on IMDb as of July 29, 2016. It says that the source is complete for the cast and crew of the movie *Reservoir Dogs*.

1.1 Motivating Scenario

We now motivate the need for formalizing and expressing completeness statements in a machine-readable way. Then, we show how completeness statements are useful for query answering.

Availability of Completeness Statements. We start our discussion by considering the IMDb data source, which provides movie-related information. Figure 1 shows a screenshot taken from the IMDb website; the page is about the movie *Reservoir Dogs* and 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 played 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*. So far, IMDb has around 24,000 editor-verified completeness statements about casts and crews for all titles in the database.² The availability of such statements 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 will probably prefer it to other pages since, assuming the truth of the statement, all they need is here.

The problem with such statements, expressed in natural language, is that they cannot be automatically processed, thus hindering their applicability, for instance, in query answering. 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 statements and informed the user about the completeness of the results.

While the IMDb example shows the availability of completeness statements by domain experts, other examples of Web data sources, such as OpenStreetMap³ and Wikipedia, provide completeness statements based on crowdsourcing. For Wikipedia, we have a complete list of paintings

²<http://www.imdb.com/interfaces>.

³http://wiki.openstreetmap.org/wiki/Hall_of_Fame/Streets_complete.

Template:Complete list

From Wikipedia, the free encyclopedia

This list is complete and up-to-date as of {{{1}}}.

(a)

Template documentation [view] [edit] [history] [purge]

Usage [edit]

Place `{{Complete list|June 2015}}` at the top or bottom of a section or list. The date field is open, and you can use any *date format*. This is an example of usage in an article:

This list is complete and up-to-date as of June 2015.

List of days of the week

- Sunday
- Monday
- Tuesday
- Wednesday
- Thursday https://en.wikipedia.org/wiki/Template:Complete_list
- Friday
- Saturday



Award recipients

[edit] https://en.wikipedia.org/wiki/Twenty-five_Year_Award

The "Year awarded" column states the year the award was handed out, and has a link to an article about the significant architectural events of that year.

This list is complete and up-to-date as of June 2014.

(b)


Year awarded	Building(s) city	Image	Architect(s)
1969	Rockefeller Center New York City		Reinhard & Hofmeister; Corbett, Harrison & MacMurray
1971	Crow Island School Winnetka, Illinois		Perkins, Wheeler & Will; Eiel & Eero Saarinen

Fig. 2. A list template for complete information with timestamps on Wikipedia (a) and its usage to state the completeness of the list of the *Twenty-five Year Award* recipients (b).

attributed to *Edvard Munch*⁴ and a complete list of Olympic medalists in archery.⁵ In fact, searching for pages on Wikipedia containing the keywords “complete list of” and “list is complete” gives around 14,000 results. If such completeness statements were exploited by machines, one would expect that there would be an incentive to publish many more of them.

Completeness Statements with Timestamps. The notion of completeness discussed so far is time-agnostic; it allows one to specify only whether (a portion of) a data source is complete. The expression of completeness statements relies on the judgment of the author of those statements (i.e., the maintainer of the source or an external agent). In this situation, more flexibility may be required with regard to the time validity of completeness statements. Therefore, one may be interested in having *completeness guarantees up to a certain time*. To cope with this aspect, we introduce timestamped completeness statements. In defining these statements, we were (again) inspired by real data sources, such as Wikipedia, that provide a template list for complete information with timestamps, as shown in Figure 2.

Figure 2 shows a list template taken from Wikipedia.⁶ The template allows one to specify that a list is “*complete and up-to-date as of {some specific date}*,” with this information being shown on each page where the template list is used. Such a statement differs from the previous type of statement in so far as it specifies up to what time the completeness holds. Wikipedia pages containing timestamped completeness statements range from the page of buildings that have ever won the Twenty-five Year Award⁷ (as shown in Figure 2) to the page of Italian DOP cheeses.⁸ Having timestamped completeness statements by nature requires less maintenance because the statements will still be correct even if new information (that happens after the given timestamp) appears.

⁴http://en.wikipedia.org/wiki/List_of_paintings_by_Edvard_Munch.

⁵http://en.wikipedia.org/wiki/List_of_Olympic_medalists_in_archery.

⁶https://en.wikipedia.org/wiki/Template:Complete_list.

⁷https://en.wikipedia.org/wiki/Twenty-five_Year_Award.

⁸https://en.wikipedia.org/wiki/List_of_Italian_DOP_cheeses.

```

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

# This RDF document provides a completeness statement about DBpedia.

dv:dbpdataset rdf:type          void:Dataset.

dv:dbpdataset c:hasComplStmt dv:st1.

dv:st1        c:hasPattern      [c:subject  [c:varName "m"];
                                c:predicate rdf:type;
                                c:object    s:Movie].

dv:st1        c:hasPattern      [c:subject  [c:varName "m"];
                                c:predicate s:director;
                                c:object    dbp:Tarantino].

dv:st1        rdfs:comment      "Complete for all Tarantino movies".

```

Fig. 3. An example of a completeness statement about DBpedia.

Machine-Readable Statements. Due to the openness characteristic of data sources made available as Linked Data, the ability to express completeness statements is a valuable aspect. The machine-readable nature of the RDF enables one to deal with the problems discussed in the example about IMDb. This is because completeness statements themselves can be represented in the RDF. As an example, the description of an RDF data source such as DBpedia⁹ (i.e., an RDF counterpart of Wikipedia) could include the meta-information that it is complete for all Quentin Tarantino movies. Figure 3 shows how DBpedia can be complemented with completeness statements expressed in our formalism. Here, we provide a high-level presentation of the completeness framework; details on the theoretical framework supporting it appear in later sections.

A completeness statement can be thought of as a SPARQL Basic Graph Pattern (BGP). The BGP

$$(?m \text{ rdf:type } s:\text{Movie}).(?m \text{ s:director } dbp:\text{Tarantino}),$$

for instance, expresses the completeness of all movies directed by Tarantino, that is, all such movies appear in the data source. In the figure, this information is represented using our completeness vocabulary (see Section 2.2). The RDF modeling of completeness statements is inspired by SPIN,¹⁰ a SPARQL inferencing notation. Furthermore, we also formalize completeness statements with timestamps and develop an RDF modeling for those in Section 5.

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 article, we will focus on how to leverage completeness statements for query answering. The research question that we address is how to assess whether available data sources with different degrees of completeness can ensure the completeness of query answers. Consider the scenario depicted in Figure 4, in which the data sources DBpedia and LinkedMDB (Hassanzadeh and Consens 2009), an RDF data source about movies, 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 completeness of his query can

⁹<http://dbpedia.org/>.

¹⁰<http://spinrdf.org/sp.html>.

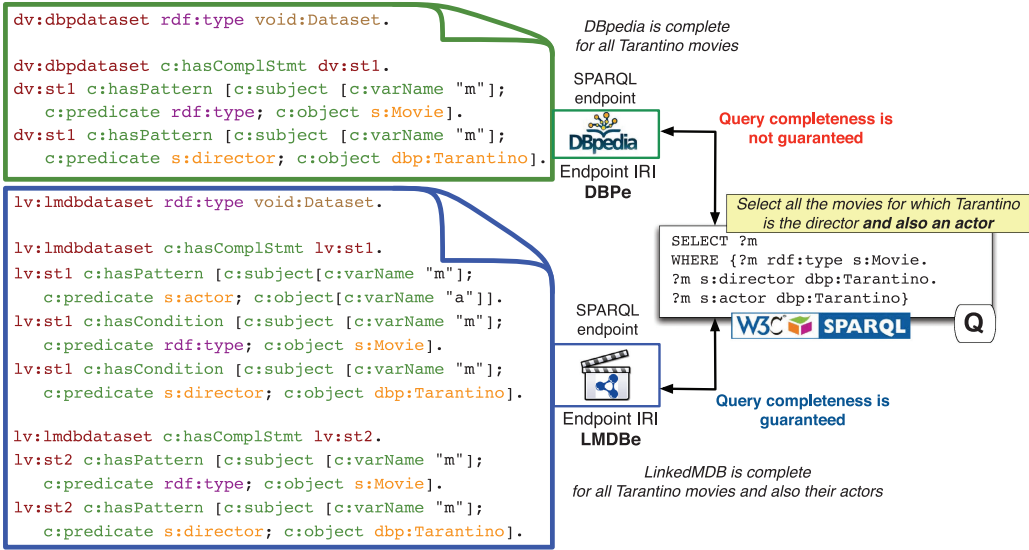


Fig. 4. Completeness statements and their usage for query answering.

be guaranteed. For instance, although DBpedia is complete for all of Tarantino movies, nothing can be said about his participation as an actor in these movies (which is required in the query). At the time of writing this article, DBpedia is actually incomplete; this is because, in the DBpedia description of the movie *Reservoir Dogs*, the fact that Tarantino played the Mr. Brown character is missing (and, from Figure 1, we know that this is the case). On the other hand, LinkedMDB can provide a complete answer. With our framework, it is possible to express in the RDF the completeness statement available in natural language in Figure 1 and infer the completeness of query answers for given completeness statements.

In this case, LinkedMDB can guarantee the completeness of the query answer because it has all the actors in Tarantino movies (represented by the statement `lv:st1`) in addition to Tarantino movies themselves (represented by the statement `lv:st2`). Note that the statement `lv:st1` consists of two parts: (i) the *pattern*, which is expressed via the BGP $(?m, s:actor, ?a)$ and (ii) the *condition*, that is, the BGP $(?m, rdf:type, s:Movie).(?m, s:director, dbp:Tarantino)$. A completeness statement allows one to say that a certain part (i.e., with respect to some condition) of data is complete or, in other words, to state that a data source contains all triples of a pattern P_1 that satisfy a condition P_2 . The detailed explanation and the semantics of completeness statements can be found in Section 2.

In the case of completeness statements with timestamps, query completeness must be approached differently. For this reason, we introduce the notion of the *guaranteed completeness date* of a query, that is, the latest date for which complete query results are guaranteed. In Section 5.1, we present motivating scenarios for guaranteed completeness dates.

Beyond Query Answering. While query answering is our focus in this article, completeness statements can be applied in various other scenarios. *Data collection:* Completeness statements are particularly useful for managing data collections, such as works of an artist, cities in countries, election results, census data, and so forth. When data collectors become aware of data completeness, they know where to focus their efforts in completing data. Moreover, their data collection efforts would get rewarded more from the use of quality stamps like completeness statements whenever they

achieve some progress on completing data. *Completeness analytics*: How complete is an entity compared to other similar entities? Which properties are mostly complete for what types of entities? Having a machine-readable RDF representation of completeness information enables analytics for answering such questions. *Search optimization*: A user wants to look for movies by Tarantino in 2008. By having completeness statements on IMDb about these movies, a search engine could stop after finding this specific source without the need to consult other sources.

1.2 Contributions

The original version of our ideas on completeness management for RDF data sources was published in the *Proceedings of the International Semantic Web Conference* (Darari et al. 2013). The present article significantly extends the previous work in the following ways:

- (1) we introduce time into completeness statements and query completeness (Section 5);
- (2) we formulate the constant-relevance principle to rule out irrelevant completeness statements in completeness reasoning and develop a retrieval technique based on this principle (Section 6.1);
- (3) we perform a comprehensive experimental evaluation about reasoning with large sets of completeness statements (Section 6.2 and Section 6.3);
- (4) we include the proofs of all theorems, as well as more recent related work, and improve the accessibility of the theoretical parts by providing many more explanations and examples.

The framework that we are going to introduce provides the necessary underpinning to complement, with *qualitative* descriptions, existing proposals such as VoID (Alexander et al. 2011) that mainly deal with *quantitative* descriptions.

1.3 Organization of the Article

The rest of the article is organized as follows. Section 2 provides some background about the RDF and SPARQL and introduces a formalization of the completeness problem for RDF data sources. This section also describes how completeness statements can be represented in RDF. Section 3 discusses how completeness statements can be used in query answering when considering a single data source at a time. In Section 4, we study query completeness for federated data sources. In Section 5, we introduce an extension of our completeness framework to deal with time. Section 6 introduces a retrieval technique to filter out irrelevant completeness statements during completeness reasoning and then discusses an experimental evaluation about completeness reasoning with large sets of statements. We present related work in Section 7. Section 8 provides a discussion of our completeness framework, while Section 9 presents conclusions and future work. We provide proofs in an electronic appendix.

2 FORMAL FRAMEWORK

We now give an overview of the RDF and SPARQL and formalize the basic notions of our completeness management framework.

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 (Klyne and Carroll 2004). For simplicity, we omit namespaces for the abstract representation of RDF graphs. In this work, we mainly focus on ground RDF graphs (i.e., no blank nodes) and discuss the role of blank nodes in Section 8.

The standard query language for the RDF is SPARQL. The basic building blocks of a SPARQL query are *triple patterns*, which resemble RDF triples, except that in each position variables are also allowed. SPARQL queries include graph patterns, built using the AND operator, and more sophisticated operators, including OPT (for “optional”). We concentrate on both the AND and OPT operators for the following reasons: (i) AND forms the core of SPARQL queries and (ii) OPT is the most distinctive feature of SPARQL with regard to classical relational database technology (Kaminski and Kostylev 2016) that is used substantially in practice (Picalausa and Vansummeren 2011; Saleem et al. 2015). Triple patterns connected with only the AND operator are called *basic graph patterns* (BGPs). Alternatively, one may use a set of triple patterns to represent a BGP. A *mapping* μ from variables to terms is defined as a partial function $\mu : V \rightarrow I \cup L$. The operator $dom(\mu)$ returns the set of all mapped variables in μ . Given two mappings μ and μ' , we say that the mappings are compatible, written $\mu \sim \mu'$, if both map the overlapping variables to the same term. The operator $var(t)$ (and $var(P)$) denotes the set of variables occurring in a triple pattern t (and a BGP P). Given a BGP P , the expression μP returns a graph in which all the variables in P are replaced with terms according to μ . Evaluating a graph pattern P over an RDF graph G results in a set of mappings from $var(P)$ to terms, denoted as $\llbracket P \rrbracket_G$. The join and left outer join between sets of mappings M and M' are defined as follows:

$$\begin{aligned} M \bowtie M' &= \{ \mu \cup \mu' \mid \mu \in M, \mu' \in M', \text{ and } \mu \sim \mu' \} \\ M \bowtie\! \! \! \bowtie M' &= (M \bowtie M') \cup \{ \mu \in M \mid \forall \mu' \in M' : \mu \not\sim \mu' \}. \end{aligned}$$

The evaluation of a SPARQL graph pattern is defined recursively as follows (Pérez et al. 2009):

$$\begin{aligned} \llbracket t \rrbracket_G &= \{ \mu \mid dom(\mu) = var(t) \text{ and } \mu(t) \in G \} \\ \llbracket P_1 \text{ AND } P_2 \rrbracket_G &= \llbracket P_1 \rrbracket_G \bowtie \llbracket P_2 \rrbracket_G \\ \llbracket P_1 \text{ OPT } P_2 \rrbracket_G &= \llbracket P_1 \rrbracket_G \bowtie\! \! \! \bowtie \llbracket P_2 \rrbracket_G, \end{aligned}$$

where t ranges over all triple patterns, and P_1 and P_2 range over all graph patterns with AND and OPT operators.¹¹

SPARQL queries come as DESCRIBE, SELECT, ASK, or CONSTRUCT queries. DESCRIBE queries, whose purpose is to describe RDF resources, are not considered in this work owing to their arbitrary nature (i.e., up to the SPARQL endpoint maintainer to decide which useful description is returned). The operator $\pi_W(\mu)$ (and $\pi_W(\Omega)$) denotes the projection of a mapping μ (and a set of mappings Ω) into the set of variables W . A SELECT query has the abstract form (W, P) , where P is a graph pattern and W is a subset of the variables in $var(P)$. A SELECT query $Q = (W, P)$ is evaluated over a graph G by projecting the mappings in $\llbracket P \rrbracket_G$ to the variables in W , written as $\llbracket Q \rrbracket_G = \pi_W(\llbracket P \rrbracket_G)$. Syntactically, an ASK query is a special case of a SELECT query in which W is empty. For an ASK query Q , we write also $\llbracket Q \rrbracket_G = \text{true}$ if $\llbracket P \rrbracket_G \neq \emptyset$, and $\llbracket Q \rrbracket_G = \text{false}$ 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 article, we only use CONSTRUCT queries where P_2 is also a BGP. The result of evaluating $Q = (P_1, P_2)$ over G is the graph $\llbracket Q \rrbracket_G$, which is obtained by instantiating the pattern P_1 with all the mappings in $\llbracket P_2 \rrbracket_G$. Further information about SPARQL can be found in Pérez et al. (2009).

Later, we will distinguish between three classes of queries: (i) *Basic queries*, that is, queries (W, P) , where P is a BGP and that return *bags* of mappings (as is the default in SPARQL); (ii) *DISTINCT queries*, that is, queries $(W, P)^d$, where P is a BGP and that return *sets* of mappings; and (iii) *OPT queries*, that is, queries $(var(P), P)$, where P is a graph pattern with OPT. Furthermore,

¹¹For the fragment in our article, the semantics also complies with the W3C semantics, as in Angles and Gutierrez (2008).

our work also considers two SPARQL evaluation extensions: SPARQL evaluation under *RDFS semantics* and *federated* SPARQL evaluation. We will discuss these in more detail in later sections.

2.1 Completeness Statements

To tackle completeness management for RDF data sources, we proceed in two steps: (i) we formalize a mechanism allowing one to specify which parts of a data source are complete and (ii) we devise techniques to check whether a query is complete over a potentially incomplete data source.

We first define completeness statements to capture which information is complete.

Definition 2.1 (Completeness Statement). A completeness statement $Compl(P_1 \mid P_2)$ consists of a nonempty BGP P_1 and a BGP P_2 . We call P_1 the *pattern* and P_2 the *condition* of the completeness statement.

We use BGPs for their flexibility to represent complex completeness information that needs more than one triple pattern and can be of various shapes (e.g., star, path, tree, cycle). Note that, by construction, completeness statements have triple granularity. As an illustration, to express that a source is complete for all pairs of triples that say “*?m is a movie and ?m is directed by Tarantino*” we use

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

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

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

whose pattern matches triples about actors and whose condition restricts the actors to those of movies directed by Tarantino. The condition in C_{act} does not imply that the data source contains triples of the form $(?m, a, Movie)$ and $(?m, director, tarantino)$. If we move the condition to the pattern, however, we impose that the data source contains the triples.

Now to model the Open World Assumption (OWA) of RDF data sources, we define an incomplete data source.

Definition 2.2 (Incomplete Data Source). We identify data sources with RDF graphs. Then, adapting a notion introduced by Motro (1989), 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.

Here, an available graph is the graph currently stored (the data that we can access), while an ideal graph is a possible conceptualization of the world. A restriction of the set of ideal graphs corresponds to an increase of knowledge about the world. Thus, by requiring that ideal graphs contain a superset of the triples in the available graph, we assume that the information represented by the available graph is correct. Without completeness statements, any graph extending the available graph can be an ideal graph. Having completeness statements restricts the possible shape of ideal graphs: regarding the parts captured by completeness statements, they must not contain more information than that stored in the available graph.

Consider, for example, a graph with two triples as the available graph: $(obama, child, malia)$, $(obama, child, sasha)$. The information about the world contained in this graph is vastly incomplete, except that it is complete for the children of Barack Obama. In the above perspective, stating that the example graph is complete for the children of Barack Obama amounts to requiring that an ideal graph modeling all information about the world, whatever its concrete shape, cannot contain further children of Obama. In Section 2.4, we will see that conclusions about query completeness are drawn from the restrictions imposed on all conceivable ideal graphs. Such reasoning, as is typical in logic, is never concerned with individual ideal graphs but rather considers the entirety

of ideal graphs that, together with the available graph, satisfy the completeness statements. Ideal graphs are the vehicle to provide a logical semantics to completeness statements. They are hypothetical in nature and neither data providers nor consumers need to deal with them.

Let us now formalize completeness statements. To a statement $C = Compl(P_1 \mid P_2)$, we associate the CONSTRUCT query $Q_C = (P_1, P_1 \text{ AND } P_2)$. Note that, over a graph G , the query Q_C returns a graph consisting of those instantiations of the pattern P_1 present in G for which the condition P_2 also can be satisfied. For example, the query $Q_{C_{act}}$ returns the cast of the Tarantino movies in graph G . We now define the semantics of completeness statements.

Definition 2.3 (Satisfaction of Completeness Statements). An incomplete data source $\mathcal{G} = (G^a, G^i)$ satisfies the statement C , written $\mathcal{G} \models C$, if $\llbracket Q_C \rrbracket_{G^i} \subseteq G^a$ holds.

Intuitively, an incomplete data source (G^a, G^i) satisfies a completeness statement C if the subgraph of G^i captured by C is also present in G^a . This definition naturally extends to the satisfaction of a set C of completeness statements, that is, $\mathcal{G} \models C$ iff for all $C \in C$, it is the case that $\llbracket Q_C \rrbracket_{G^i} \subseteq G^a$.

Example 2.4. Consider the DBpedia data source that contains information about Tarantino-related movies:

$$G_{dbp}^a = \{(reservoirDogs, director, tarantino), (pulpFiction, director, tarantino), \\ (killBill, director, tarantino), (desperado, actor, tarantino), \\ (pulpFiction, actor, tarantino), (desperado, a, Movie), \\ (reservoirDogs, a, Movie), (pulpFiction, a, Movie), (killBill, a, Movie)\}.$$

A possible extension of the above graph is the graph G_{dbp}^i , which also contains the information that Tarantino starred in Reservoir Dogs:¹²

$$G_{dbp}^i = G_{dbp}^a \cup \{(reservoirDogs, actor, tarantino)\}.$$

Putting the above two graphs together forms the incomplete data source $\mathcal{G}_{dbp} = (G_{dbp}^a, G_{dbp}^i)$. In this case, the statement C_{dir} (Equation (1)) is satisfied by \mathcal{G}_{dbp} , since all triples from evaluating $Q_{C_{dir}}$ over G_{dbp}^i are included in G_{dbp}^a . In contrast, the statement C_{act} (Equation (2)) is not satisfied by \mathcal{G}_{dbp} because $Q_{C_{act}}$ returns over G_{dbp}^i the triple $(reservoirDogs, actor, tarantino)$ that is not in G_{dbp}^a .

An important tool for characterizing completeness entailment in the next sections is the *transfer operator* T_C , which captures the complete parts of a graph with regard to a set of completeness statements. Given a set C of completeness statements and a graph G , the operator is defined as

$$T_C(G) = \bigcup_{C \in C} \llbracket Q_C \rrbracket_G. \quad (3)$$

The operator takes the union of evaluating over G all the corresponding CONSTRUCT queries of the statements in C . In terms of incomplete data sources, the transfer operator takes the parts of the ideal graph that have to be present in the available graph. In a way, it *transfers* complete information from the ideal graph to the available graph. Crucial properties of the transfer operator are summarized in the following proposition, which follows directly from the construction of T_C and the definition of the satisfaction of C .

¹²Which is actually the case in the real world.

PROPOSITION 2.5 (PROPERTIES OF T_C). *Let C be a set of completeness statements. Then,*

$$(1) (G^a, G^i) \models C \text{ iff } T_C(G^i) \subseteq G^a.$$

Consequently, for any graph G , we have that

- (2) *the pair $(T_C(G), G)$ is an incomplete data source satisfying C , and*
- (3) *$T_C(G)$ is the smallest available graph for which C holds.*

2.2 RDF Representation of Completeness Statements

Practically speaking, completeness statements should be compliant with the existing ways of providing metadata about RDF data sources, for instance, by enriching current proposals such as VoID (Alexander et al. 2011). Hence, it becomes essential to be able to express completeness statements in RDF itself. Suppose that we want to express that LinkedMDB satisfies the following completeness statement (as in Equation (2)):

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

To reach this goal, we need (i) a vocabulary to say that this is a statement about LinkedMDB; (ii) a mechanism to state which triple patterns form the statement's pattern and which form the statement's condition; and (iii) a mechanism to represent the constituents of the triple patterns, namely, the subject, predicate, and object of a triple pattern. We introduce the following property names whose meaning is intuitive:

hasComplStmt, hasPattern, hasCondition, subject, predicate, object.

If a constituent of a triple pattern is a term (an IRI or a literal), then it can be specified directly in RDF. As this is not possible for variables, we represent a variable by a resource that has a literal value for the property varName. In light of these considerations, we can represent C_{act} in RDF as the resource lv:st1 described in Figure 4.

More generally, consider a completeness statement $Compl(P_1 \mid P_2)$, where $P_1 = \{t_1, \dots, t_n\}$ and $P_2 = \{t_{n+1}, \dots, t_m\}$ 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 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. Our vocabulary is available at <http://completeness.inf.unibz.it/ns>.

2.3 Query Completeness

The standard way to access data is via queries. When querying a data source, we want to know whether the data source provides all the information needed to answer the query, that is, whether the query is complete with regard to the real world. For instance, when querying DBpedia for movies directed by Tarantino, it would be interesting to know whether we really get *all* such movies. Intuitively, over an incomplete data source, a query is complete whenever all answers we retrieve over the ideal graph coincide with those over the available graph.

Definition 2.6 (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 $Compl(Q)$ if Q returns the same result over G^a as it does over G^i , that is, $\llbracket Q \rrbracket_{G^a} = \llbracket Q \rrbracket_{G^i}$. In this case, we write $\mathcal{G} \models Compl(Q)$.

In this work, we focus on classes of queries that are monotonic. Therefore, by definition, it holds that $\llbracket Q \rrbracket_{G^a} \subseteq \llbracket Q \rrbracket_{G^i}$ for all incomplete data sources $\mathcal{G} = (G^a, G^i)$. We note that without monotonicity the problem of query soundness might also arise. Dealing with this problem is beyond the scope of this article.

Example 2.7. Consider the incomplete data source \mathcal{G}_{dbp} as in Example 2.4 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:

$$Q_{dir} = (\{?m\}, \{(?m, a, Movie), (?m, director, tarantino)\})$$

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

Then, it holds that Q_{dir} is complete over \mathcal{G}_{dbp} since $\llbracket Q_{dir} \rrbracket_{G_{dbp}^a} = \{\{?m \mapsto reservoirDogs\}, \{?m \mapsto pulpFiction\}, \{?m \mapsto killBill\}\} = \llbracket Q_{dir} \rrbracket_{G_{dbp}^i}$. On the other hand, $Q_{dir+act}$ is *not* complete over \mathcal{G}_{dbp} since $\llbracket Q_{dir+act} \rrbracket_{G_{dbp}^a}$ does not have the mapping $\{?m \mapsto reservoirDogs\}$, which is in $\llbracket Q_{dir+act} \rrbracket_{G_{dbp}^i}$.

2.4 Completeness Entailment

From the notions above, a question arises related to when some meta-information about data completeness can provide a guarantee for query completeness. The question is whether the completeness statements guarantee that the available state already contains all data that is required for computing the query answer so that one can trust the results of the query. In the following, we formalize the entailment of query completeness by completeness statements. While previously we have looked at examples of concrete incomplete data sources, we now “quantify” *all* incomplete data sources satisfying the statements, requiring that if an incomplete data source satisfies the completeness statements, then it must also satisfy the query completeness.

Definition 2.8 (Completeness Entailment). Let C be a set of completeness statements and Q be a SELECT query. We say that C *entails the completeness of* Q , written $C \models Compl(Q)$, if every incomplete data source that satisfies C also satisfies $Compl(Q)$.

Checking whether completeness entailment holds is the core problem on which we will focus in the rest of the article. Since entailment, by universal quantification, involves *all* incomplete data sources satisfying C , and since for each available graph G^a there can be many, usually infinitely many, ideal graphs G^i such that $(G^a, G^i) \models C$, reasoning about completeness does not depend on any specific ideal graph G^i but rather on the statements in C .

Example 2.9. Consider C_{dir} from Equation (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 the query Q_{dir} from Example 2.7. Thus, $\{C_{dir}\} \models Compl(Q_{dir})$. However, this is not enough to completely answer $Q_{dir+act}$; thus, $\{C_{dir}\} \not\models Compl(Q_{dir+act})$. We will see later how this intuitive reasoning can be formalized in various settings.

3 COMPLETENESS REASONING OVER A SINGLE DATA SOURCE

In this section, we show *how* completeness statements can be used to judge whether a query returns a complete answer. We study this problem in different settings: (i) completeness statements that hold over a *single* data source; (ii) completeness statements in a *federation* of data sources (see Section 4); and (iii) completeness statements with *timestamps* (see Section 5). For completeness statements over a single data source, we devise characterizations of completeness entailment for basic queries, DISTINCT queries, OPT queries, and queries under RDFS semantics.

3.1 Completeness Entailment for Basic Queries

One of the query classes that we consider in this work is the class of queries with a conjunctive body. The standard semantics for such queries is bag semantics, which allows repetition of results. Generally, a basic query Q is complete with regard to a set C of completeness statements, if for

every incomplete data source $\mathcal{G} = (G^a, G^i)$ satisfying C , the query answers over G^i are contained in the query answers over G^a , where *duplicates are taken into account*. That is, a mapping occurring n times in $\llbracket Q \rrbracket_{G^i}$ must occur at least n times in $\llbracket Q \rrbracket_{G^a}$. Actually, since $G^a \subseteq G^i$ and since conjunctive queries are monotonic, we always have that the bag $\llbracket Q \rrbracket_{G^a}$ is contained in the bag $\llbracket Q \rrbracket_{G^i}$. Hence, Q is complete over \mathcal{G} iff every mapping occurring n times in $\llbracket Q \rrbracket_{G^i}$ occurs also n times in $\llbracket Q \rrbracket_{G^a}$.

We want to provide a characterization of completeness entailment for basic queries. Let us give an example to provide an intuition of the characterization.

Example 3.1. Consider the set $C_{dir,act}$ consisting of C_{dir} from Equation (1) and C_{act} from Equation (2). Recall the query $Q_{dir+act} = (\{ ?m \}, P_{dir+act})$, where

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

We want to check whether these statements entail the completeness of $Q_{dir+act}$, that is, whether $C_{dir,act} \models Compl(Q_{dir+act})$ holds.

Suppose that $\mathcal{G} = (G^a, G^i)$ satisfies $C_{dir,act}$. 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 $(m', a, Movie)$, $(m', director, tarantino)$, and $(m', actor, tarantino)$. Intuitively, these triples constitute a graph that is prototypical for the query, since any graph over which the query returns an answer m' , contains a subgraph of this shape.

The CONSTRUCT query $Q_{C_{dir}}$, corresponding to our first completeness statement, returns over the graph $\mu P_{dir+act}$ the two triples $(m', a, Movie)$ and $(m', director, tarantino)$, while the CONSTRUCT query $Q_{C_{act}}$, corresponding to the second completeness statement, returns the triple $(m', actor, tarantino)$. Thus, all triples in $\mu P_{dir+act}$ are being reconstructed by $T_{C_{dir,act}}$ from $\mu P_{dir+act}$.

Now, we have that

$$\mu P_{dir+act} = T_{C_{dir,act}}(\mu P_{dir+act}) \subseteq T_{C_{dir,act}}(G^i) \subseteq G^a,$$

where the last inclusion holds due to $\mathcal{G} \models C_{dir,act}$. Therefore, our query $Q_{dir+act}$ returns the mapping μ also over G^a . Since μ and \mathcal{G} were arbitrary, this shows that $C_{dir,act} \models Compl(Q_{dir+act})$ holds. Equivalently, for every incomplete data source $\mathcal{G} = (G^a, G^i)$ satisfying $C_{dir,act}$, we have that $\llbracket Q_{dir+act} \rrbracket_{G^a} = \llbracket Q_{dir+act} \rrbracket_{G^i}$.

In summary, in this example, we have reasoned about a set of completeness statements C and a basic 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 prototypical graph μP . Then, we have verified that $\mu P = T_C(\mu P)$. From this, we could conclude that, for every incomplete data source $\mathcal{G} = (G^a, G^i)$ satisfying C , we have that $\llbracket Q \rrbracket_{G^a} = \llbracket Q \rrbracket_{G^i}$. Next, we make this approach formal.

Definition 3.2 (Prototypical Graph). Let (W, P) be a query. The *freeze mapping* \tilde{id} is defined as mapping each variable $?v$ in $var(P)$ to a new IRI \tilde{v} . Instantiating the graph pattern P with \tilde{id} yields the 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. To check whether the completeness of a query is entailed by a set of completeness statements, we evaluate all the corresponding CONSTRUCT queries of the statements over the prototypical graph \tilde{P} and check whether over the evaluation result we have \tilde{P} back. Intuitively, this means that whenever there is an answer of the query over a possible ideal graph, the completeness statements guarantee that the available graph also has the data to return that answer.

THEOREM 3.3 (COMPLETENESS OF BASIC QUERIES). *Let C be a set of completeness statements and $Q = (W, P)$ be a basic query. Then,*

$$C \models \text{Compl}(Q) \quad \text{iff} \quad \tilde{P} = T_C(\tilde{P}).$$

The following complexity result follows from the fact that a completeness check is basically evaluating a union of conjunctive queries (recall the definition of the T_C -operator).

COROLLARY 3.4. *Deciding whether $C \models \text{Compl}(Q)$, given a set C of completeness statements and a basic query Q , is NP-complete.*

The result shows that the complexity of completeness reasoning is no higher than that of conjunctive query evaluation, which is also NP-complete (Chandra and Merlin 1977).

3.2 Completeness Entailment for DISTINCT Queries

Basic queries return bags of answers (i.e., they may contain duplicates), while DISTINCT *eliminates duplicates*. In the following, we illustrate the difference between basic and DISTINCT queries for completeness entailment.

Example 3.5 (Basic vs. DISTINCT Queries). Consider the singleton C_{os} containing the statement

$$C_{os} = \text{Compl}((?m, award, oscar) \mid \emptyset),$$

meaning that we have all triples of things winning an Oscar. Consider also the basic query

$$Q_{aw} = (W_{aw}, P_{aw}) = (\{?m\}, \{ (?m, award, oscar), (?m, award, ?aw) \}),$$

asking how many awards an Oscar-winning thing has won (as the triple pattern $(?m, award, ?aw)$ may give duplicates depending on the multiplicity of $?aw$). Based on Theorem 3.3, it follows that $C_{os} \not\models \text{Compl}(Q_{aw})$ since $\tilde{P}_{aw} \neq T_{C_{os}}(\tilde{P}_{aw})$. However, the situation changes if the query is a DISTINCT query, that is, the query is instead

$$Q_{awd} = (W_{awd}, P_{awd})^d = (\{?m\}, \{ (?m, award, oscar), (?m, award, ?aw) \})^d,$$

which only asks for things having won the Oscar (note here that the duplicates with regard to $?aw$ are removed). Now, we want to check whether $C_{os} \models \text{Compl}(Q_{awd})$ holds. Suppose that an incomplete data source $\mathcal{G} = (G^a, G^i)$ satisfies C_{os} . Suppose also that Q_{awd} returns a mapping $\nu = \{?m \mapsto m'\}$ over G^i for some term m' . Thus, there must be a mapping μ of the graph pattern P_{awd} of our query whose projection onto W_{awd} gives ν , and where $\mu P_{awd} \subseteq G^i$. Suppose that $\mu = \{?m \mapsto m', ?aw \mapsto aw'\}$ for some term aw' . Therefore, G^i contains $\mu P_{awd} = \{ (m', award, oscar), (m', award, aw') \}$.

The CONSTRUCT query $Q_{C_{os}}$, corresponding to our only completeness statement in C_{os} , returns over μP_{awd} the triple $(m', award, oscar)$. Now, we have that

$$\{ (m', award, oscar) \} = T_{C_{os}}(\mu P_{awd}) \subseteq T_{C_{os}}(G^i) \subseteq G^a,$$

where the last inclusion holds due to $\mathcal{G} \models C_{os}$.

However, even though $\mu P_{awd} \neq T_{C_{os}}(\mu P_{awd})$, our query Q_{awd} still returns the mapping ν over G^a , since for the subset $T_{C_{os}}(\mu P_{awd})$ of G^a , it is the case that $\nu \in \llbracket Q_{awd} \rrbracket_{T_{C_{os}}(\mu P_{awd})}$, as there exists a mapping $\nu' = \{?m \mapsto m', ?aw \mapsto oscar\}$ in $\llbracket P_{awd} \rrbracket_{T_{C_{os}}(\mu P_{awd})}$, which when projected onto W_{awd} gives ν . This shows that $C_{os} \models \text{Compl}(Q_{awd})$ holds because ν and \mathcal{G} were arbitrary.

For a query Q involving DISTINCT, the difference in the characterization in Theorem 3.3 of basic queries is that, instead of retrieving the full graph \tilde{P} after applying T_C , we only check whether sufficient parts of \tilde{P} are preserved that still allow us to retrieve the identity mapping on the distinguished variables of Q (recall the definitions of \tilde{P} and \tilde{id} in Definition 3.2).

THEOREM 3.6 (COMPLETENESS OF DISTINCT QUERIES). *Let C be a set of completeness statements and $Q = (W, P)^d$ be a DISTINCT query. Then,*

$$C \models \text{Compl}(Q) \quad \text{iff} \quad \pi_W(\tilde{id}) \in \llbracket Q \rrbracket_{T_C(\tilde{P})}.$$

The complexity of completeness entailment for DISTINCT queries is NP-complete, which is the same as for basic queries.

COROLLARY 3.7. *Deciding whether $C \models \text{Compl}(Q)$, given a set C of completeness statements and a DISTINCT query $Q = (W, P)^d$, is NP-complete.*

As discussed, checking the completeness of DISTINCT queries is slightly different from that of basic queries, since now we can ignore the multiplicity of mappings. Both problems are in the same complexity class and concern the conjunctive fragment of SPARQL. In the next section, we characterize completeness for the OPT fragment of SPARQL.

3.3 Completeness Entailment for OPT Queries

One interesting feature of SPARQL is the OPT (“optional”) operator. With OPT, one can specify that parts of a query are only evaluated whenever possible, just like the left outer join in SQL. For example, when querying for movies, one can also ask for the prizes that they won, *if any*. The OPT operator is used substantially in practice (Picalausa and Vansummeren 2011). Intuitively, the mappings for a pattern $(P_1 \text{ OPT } P_2)$ are computed as the union of all the mappings of P_1 that can be extended with the mappings of P_2 and those that cannot. Completeness entailment for queries with OPT differs from that of queries without, as illustrated below.

Example 3.8 (Completeness with OPT). Consider the following query with OPT:

$$Q_{maw} = (\{ ?m, ?aw \}, ((?m, a, \text{Movie}) \text{ OPT } (?m, \text{award}, ?aw))),$$

asking for all movies and, if available, their awards. Consider also the completeness statement expressing that all movies that have an award are complete and all awards of movies are complete:

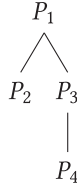
$$C_{aw} = \text{Compl}((?m, a, \text{Movie}), (?m, \text{award}, ?aw) \mid \emptyset).$$

If the query Q_{maw} used AND instead of OPT, then its completeness would be entailed by C_{aw} . With OPT in Q_{maw} , however, more completeness is required: those movies that do not have an award also have to be complete. Thus, C_{aw} alone does not entail the completeness of Q_{maw} .

If one uses OPT without restrictions, queries may display an unintended nonmonotonic behavior. Moreover, the combined complexity of the queries is PSPACE-complete (Schmidt et al. 2010). Pérez et al. (2009) have introduced the class of so-called *well-designed graph patterns*, which avoids the anomaly that may otherwise occur. Furthermore, this query class has a lower complexity, coNP-complete. Formally, a graph pattern P is *well designed* if for every subpattern $P' = (P_1 \text{ OPT } P_2)$ 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 graph patterns, which we call *well-designed OPT 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 $((N, E, r), \mathcal{P})$, where (i) (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 $\mathcal{P}(n)$. Every graph pattern can be represented by a pattern tree. As an example, consider a pattern $((P_1 \text{ OPT } P_2) \text{ OPT } (P_3 \text{ OPT } P_4))$, where P_1 to P_4 are BGPs. Its corresponding pattern tree would have the 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 . The tree can be depicted as follows:



The tree representation of well-designed graph patterns has a specific characteristic: whenever there is a common variable between two nodes in the tree, there exists a common ancestor of these nodes in which the variable also appears. Pattern trees with this property are called *quasi-well-designed pattern trees* (QWDPTs) (Letelier et al. 2012). In general, such trees have desirable properties: one can exploit the trees to be an execution plan for well-designed queries; and there exists a normal form to eliminate redundant triples and nodes in the trees, called the NR normal form. Any well-designed graph pattern P can be translated into an equivalent QWDPT in NR normal form in polynomial time (Letelier et al. 2012).

For every node n in \mathcal{T} , we define the branch pattern P_n of n as the set union of the BGPs along the path from n to the root of \mathcal{T} . Then, the *branch query* Q_n of n has the form $(\text{var}(P_n), P_n)$. The following theorem characterizes the completeness of well-designed OPT queries via branch queries.

THEOREM 3.9 (COMPLETENESS OF OPT QUERIES). *Let C be a set of completeness statements, Q be a well-designed OPT query, and \mathcal{T} be an equivalent QWDPT of Q in NR normal form. Then,*

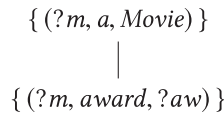
$$C \models \text{Compl}(Q) \quad \text{iff} \quad C \models \text{Compl}(Q_n) \quad \text{for all branch queries } Q_n \text{ of } \mathcal{T}.$$

Technically, the theorem allows us to reduce the problem of completeness checking for OPT queries to that of basic queries. As shown in the corollary below, the problem is still in NP (and NP-hard) because the number of guesses is basically k times more than that of a single basic query (i.e., the yes-certificate is still polynomially long), where k is the number of branch queries.

COROLLARY 3.10. *Deciding whether $C \models \text{Compl}(Q)$, given a set C of completeness statements and a well-designed OPT query Q , is NP-complete.*

In the following, we revisit Example 3.8 by applying our characterization in Theorem 3.9.

Example 3.11 (Completeness with OPT Revisited). Consider the query Q_{maw} and the completeness statement C_{aw} as above. The following is the corresponding QWDPT representation \mathcal{T} of Q_{maw} in NR normal form:



The branch queries of \mathcal{T} are the following two queries:

- $Q_{maw1} = (\{ ?m \}, \{ (?m, a, \text{Movie}) \})$ and
- $Q_{maw2} = (\{ ?m, ?aw \} \{ (?m, a, \text{Movie}), (?m, \text{award}, ?aw) \})$.

Because of Theorem 3.3, it holds that $\{ C_{aw} \} \models \text{Compl}(Q_{maw2})$, but $\{ C_{aw} \} \not\models \text{Compl}(Q_{maw1})$. Thus, from Theorem 3.9, it follows that $\{ C_{aw} \} \not\models \text{Compl}(Q_{maw})$.

3.4 Completeness Entailment Under the RDFS Semantics

RDFS is a simple ontology language widely used for RDF data (Brickley and Guha 2004). With RDFS, one can express subclass and subproperty as well as domain and range relationships. RDFS information allows for additional inferences about data and needs to be taken into account when checking completeness entailment.

Example 3.12 (RDF vs. RDFS). Consider the query

$$Q_{film} = (\{ ?m \}, \{ (?m, a, Film) \}),$$

asking for all films, and the completeness statement

$$C_{movie} = Compl((?m, a, Movie) \mid \emptyset),$$

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. Consider now the RDFS schema S_{fm} , consisting of the two axioms $(Film, subclass, Movie)$ and $(Movie, subclass, Film)$, stating that films and movies are equivalent. Taking into account the schema S_{fm} , we can, in fact, conclude that the statement C_{movie} entails the completeness of Q_{film} .

To see this, suppose that (G^a, G^i) is an incomplete data source satisfying C_{movie} , with respect to the schema S_{fm} . If the query Q_{film} retrieves a film f over the ideal graph G^i , then f is also a movie, owing to the first axiom. Then, C_{movie} implies that the movie f occurs also in the available graph G^a . Applying the second axiom, which is the converse of the first one, we infer that f is also a film in G^a . Thus, Q_{film} retrieves f also over G^a .

Note that we applied the axioms from S_{fm} twice: first, for the ideal graph, we translated films into movies to make the completeness statement applicable, and second, for the available graph, we translated movies into films to make the query applicable.

We focus on *minimal RDFS*, which formalizes the essence of RDFS (Muñoz et al. 2009), avoiding axiomatic information that only reasons about the internals of the language itself and not about the data. The minimal RDFS vocabulary contains the terms subproperty, subclass, domain, range, and type. A *schema graph* S is a set of triples built using any of the minimal RDFS terms, except type, as predicates. As a shortcut, we may also use the property a to represent type information.

We assume that schema information is not lost in incomplete data sources. Hence, for incomplete data sources, it is possible to extract their RDFS schema into a separate graph. The *closure of a graph* G with regard to a schema S , written $cl_S(G)$, 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 nonschema triples as $cl_S(G) = cl(S \cup G)$. Below, we show the derivation rules for computing the closure (Muñoz et al. 2009), where sp is short for subproperty, sc for subclass, dom for domain, rng for range.

$$\frac{(?a, sp, ?b) (?b, sp, ?c)}{(?a, sp, ?c)} \quad \frac{(?a, sc, ?b) (?b, sc, ?c)}{(?a, sc, ?c)} \quad \frac{(?a, dom, ?b) (?x, ?a, ?y)}{(?x, type, ?b)}$$

$$\frac{(?a, sp, ?b) (?x, ?a, ?y)}{(?x, ?b, ?y)} \quad \frac{(?a, sc, ?b) (?x, type, ?a)}{(?x, type, ?b)} \quad \frac{(?a, rng, ?b) (?x, ?a, ?y)}{(?y, type, ?b)}$$

We now say that a set C of completeness statements *entails* the completeness of a query Q with regard to an RDFS schema graph S , written $C \models_S Compl(Q)$, if for all incomplete data sources (G^a, G^i) it holds that, if the pair $(cl_S(G^a), cl_S(G^i))$ satisfies C , then the pair also satisfies $Compl(Q)$.

An essential tool to perform completeness reasoning is the transfer operator. To take account of the role of the schema, we modify the transfer operator in such a way that we first compute the schema closure of the argument, then transfer the triples, and, finally, compute the schema closure of the result. In this way, we consider the RDFS closure of the data graph, and then also

the RDFS inferences of the “transferred” data with regard to the completeness statements. For a set of completeness statements C and a schema graph S , let T_C^S denote the function composition $cl_S \circ T_C \circ cl_S$. Then, the following theorem holds.

THEOREM 3.13 (COMPLETENESS UNDER RDFS). *Let C be a set of completeness statements, $Q = (W, P)$ be a basic query, and S be a schema graph. Then,*

$$C \models_S \text{Compl}(Q) \quad \text{iff} \quad \tilde{P} \subseteq T_C^S(\tilde{P}).$$

The complexity of completeness entailment with RDFS is still NP-complete due to the tractability of the RDFS closure computation (Muñoz et al. 2009).

COROLLARY 3.14. *Deciding whether $C \models_S \text{Compl}(Q)$, given a set C of completeness statements, a schema graph S , and a basic query Q , is NP-complete.*

An analogous characterization holds for checking the completeness of DISTINCT queries under RDFS semantics. The following example shows how the theorem is used to check the completeness of a query under RDFS semantics.

Example 3.15 (Completeness under RDFS). Consider the query Q_{film} , the statement C_{movie} , and the schema graph S_m , as above. From Theorem 3.3, we have that $\{C_{movie}\} \not\models \text{Compl}(Q_{film})$. However, we have that $\tilde{P}_{film} = \{(\tilde{m}, a, Film)\} \subseteq \{(\tilde{m}, a, Film), (\tilde{m}, a, Movie)\} = T_{\{C_{movie}\}}^S(\tilde{P}_{film})$. Thus, by Theorem 3.13, it holds that $\{C_{movie}\} \models_{S_m} \text{Compl}(Q_{film})$.

4 COMPLETENESS REASONING OVER FEDERATED DATA SOURCES

Data on the Web is inherently distributed. Hence, the single-source query mechanism of SPARQL has been extended to deal with multiple data sources (Prud’hommeaux and Buil-Aranda 2013). This extension allows us to express a *federated query*, which is a SPARQL query evaluated across diverse data sources using the SERVICE keyword. In particular, in the body of a federated query, it is possible to specify which parts of the query are to be executed at which SPARQL endpoints of the data sources.

Up to this point, we have studied the problem of querying a single data source annotated with completeness statements. The federated scenario calls for an extension of the completeness framework discussed in Section 3 to guarantee query completeness based on completeness statements over multiple data sources. In this setting, as opposed to a single available graph, we now have multiple, federated available graphs, each of which may be complete for different aspects of the data. Moreover, there is just (hypothetically) one ideal graph to reflect the idea that the ideal graph captures all the facts of the real world, while each available graph contains parts of those facts. Given such a federation extension, the question we want to tackle is as follows: given a nonfederated query (i.e., a query without the SERVICE keyword) and completeness statements over multiple data sources, how can the query be *rewritten* into a federated version such that each query part is assigned to a suitable source that can guarantee the query part’s completeness, hence ensuring the completeness of the whole query? We call such a rewriting a *smart rewriting*. This section discusses how we develop the federation extension to address this question.

Federated SPARQL Queries. Before introducing the federation extension of the completeness management framework, we first formalize the notion of federated SPARQL queries. A federated SPARQL query is a SPARQL query that contains the SERVICE keyword. A federated SPARQL query is executed over a federated graph. Formally, a *federated graph* is a family (Barile 2016) of RDF graphs $\tilde{G} = (G_j)_{j \in J}$, where J is a set of IRIs of data sources (or their endpoints). A federated SPARQL query (as for the case of a nonfederated query) can be a SELECT or an ASK query (Arenas et al. 2010). In what follows, we focus on the conjunctive fragment (i.e., the AND fragment) of SPARQL with

the inclusion of the SERVICE keyword. In addition, we assume that each triple in a federated query occurs within the scope of a (nonvariable) SERVICE call and that endpoint IRIs in the set J always resolve to their corresponding graphs. Thus, for presentation purposes, we provide a simpler, but compliant, semantics to the existing SERVICE semantics (Buil-Aranda et al. 2013).

As opposed to nonfederated SPARQL queries, which are evaluated over graphs, in the federated scenario, queries are evaluated over a pair (i, \tilde{G}) , where the first component is an IRI indicating the current SPARQL endpoint, and the second component is a federated graph. The semantics of graph patterns with AND and SERVICE operators is defined as follows:

$$\begin{aligned} \llbracket t \rrbracket_{(i, \tilde{G})} &= \llbracket t \rrbracket_{G_i} \\ \llbracket P_1 \text{ AND } P_2 \rrbracket_{(i, \tilde{G})} &= \llbracket P_1 \rrbracket_{(i, \tilde{G})} \bowtie \llbracket P_2 \rrbracket_{(i, \tilde{G})} \\ \llbracket (\text{SERVICE } j P) \rrbracket_{(i, \tilde{G})} &= \llbracket P \rrbracket_{(j, \tilde{G})}, \end{aligned}$$

where t ranges over all triple patterns, and $P, P_1,$ and P_2 range over all graph patterns with AND and SERVICE operators. We denote federated queries as \tilde{Q} . Note that, for a federated query \tilde{Q} , the result $\llbracket \tilde{Q} \rrbracket_{(i, \tilde{G})}$ is independent of the “default” endpoint i , since we assume that each triple in \tilde{Q} occurs within the scope of a SERVICE call that specifies where to evaluate the triple.

The following will be the running example of this section.

Example 4.1. Consider the two data sources of DBpedia and LMDb shown in Figure 4 plus an additional data source named FB (Facebook), whose endpoints are reachable at the IRIs DBPe, LMDBe, and FBe, respectively. A federated query \tilde{Q}_{fb} that asks about Tarantino movies over LMDb and their number of likes over FB is shown below:

$$\begin{aligned} \tilde{Q}_{fb} = (&\{?m, ?l\}, ((\text{SERVICE LMDBe } \{(?m, a, \text{Movie}), (?m, \text{director}, \text{tarantino})\}) \text{ AND} \\ &(\text{SERVICE FBe } \{(?m, \text{likes}, ?l)\}))). \end{aligned}$$

4.1 Federated Completeness Framework

We now extend our completeness management framework to the federated setting. We assume from now on that the set J of IRIs of data sources is fixed and all indexes are drawn from J . We first formalize incomplete federated data sources that compare multiple available data sources with a single ideal source.

Definition 4.2 (Incomplete Federated Data Source). An *incomplete federated data source* (incomplete FDS) is a pair $\tilde{\mathcal{G}} = (\tilde{G}^a, G^i)$ consisting of an *available federated graph* $\tilde{G}^a = (G_j^a)_{j \in J}$ and an *ideal graph* G^i , such that $G_j^a \subseteq G^i$ for all $j \in J$.

Example 4.3. Let the set of IRIs be $J = \{\text{DBPe}, \text{LMDBe}, \text{FBe}\}$. Consider an incomplete FDS $\tilde{\mathcal{G}} = (\tilde{G}^a, G^i)$ about Tarantino movies, which is graphically represented in Figure 5.

The definition of an incomplete FDS captures the intuition that the ideal graph represents all the facts that hold in the world, while each data source contains a part of those facts. Note that, assuming a global schema is used, the graphs of the sources may overlap. Next, we adapt completeness statements in the federated setting, making explicit for which specific data sources the statements are intended.

Definition 4.4 (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,

$$((G_j^a)_{j \in J}, G^i) \models_{fed} (C, k) \quad \text{iff} \quad (G_k^a, G^i) \models C.$$

This definition naturally extends to sets $\tilde{\mathcal{C}}$ of indexed completeness statements.

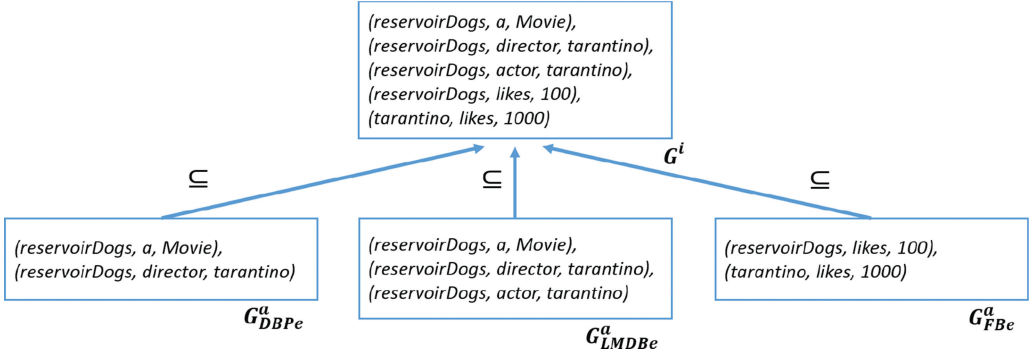


Fig. 5. An incomplete federated data source about Tarantino movies.

Example 4.5. Consider the completeness statements C_{dir} in Equation (1), C_{act} in Equation (2), and $C_{fb} = \text{Compl}((?m, likes, ?l) \mid \emptyset)$ about the completeness of the number of likes. Let \bar{C} be the set of indexed completeness statements, defined as follows:

$$\bar{C} = \{ (C_{dir}, \text{DBPe}), (C_{dir}, \text{LMDBe}), (C_{act}, \text{LMDBe}), (C_{fb}, \text{FBe}) \}.$$

One can readily check that the incomplete FDS $\bar{\mathcal{G}}$ as above satisfies the set \bar{C} .

Via flattening, we associate a nonfederated version to each federated query, federated graph, incomplete FDS, and set of indexed completeness statements.

Definition 4.6 (Flattening). We define flattenings for federated queries, federated graphs, incomplete FDSs, and sets of indexed completeness statements as follows:

- The flattening \bar{Q}^{fl} of a federated query \bar{Q} is obtained by replacing each occurrence of a SERVICE call (SERVICE $j P$) in \bar{Q} 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{C}^{fl} of a set \bar{C} of indexed completeness statements is the set $\bar{C}^{fl} = \{ C \mid (C, k) \in \bar{C} \}$, where we ignore the indexes.

In the federated setting, the notion of completeness entailment is different from that over a single source, as we now have to deal with indexed completeness statements.

Definition 4.7 (Federated Completeness and Entailment). A federated query \bar{Q} is complete over an incomplete FDS $\bar{\mathcal{G}} = (\bar{G}^a, G^i)$, written $\bar{\mathcal{G}} \models_{fed} \text{Compl}(\bar{Q})$, if $\llbracket \bar{Q} \rrbracket_{(j_0, \bar{G}^a)} = \llbracket \bar{Q}^{fl} \rrbracket_{G^i}$ for every 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{C} is a set of indexed completeness statements, then \bar{C} entails $\text{Compl}(\bar{Q})$, written $\bar{C} \models_{fed} \text{Compl}(\bar{Q})$, if $\bar{\mathcal{G}} \models_{fed} \bar{C}$ implies $\bar{\mathcal{G}} \models_{fed} \text{Compl}(\bar{Q})$ for all incomplete FDSs $\bar{\mathcal{G}}$.

Example 4.8. Consider again the incomplete FDS $\bar{\mathcal{G}}$ and the federated query \bar{Q}_{fb} . We have that $\bar{\mathcal{G}} \models_{fed} \text{Compl}(\bar{Q}_{fb})$. The reason is that for every IRI $j_0 \in J$, it holds that $\llbracket \bar{Q}_{fb} \rrbracket_{(j_0, \bar{G}^a)} = \{ \{ ?m \mapsto \text{reservoirDogs}, ?l \mapsto 100 \} \} = \llbracket \bar{Q}_{fb}^{fl} \rrbracket_{G^i}$.

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

PROPOSITION 4.9. *Let \bar{C} be a set of indexed completeness statements and Q be a basic query. Then,*

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

Example 4.10. Consider the set \bar{C} of indexed completeness statements, introduced in Example 4.5. Suppose that the nonfederated query Q_{fb} is the flattened version of \bar{Q}_{fb} . Since it is the case that $\tilde{P}_{fb} = \{ (\tilde{m}, a, Movie), (\tilde{m}, director, tarantino), (\tilde{m}, likes, \tilde{l}) \} = T_{\bar{C}^{fl}}(\tilde{P}_{fb})$, we have that $\bar{C}^{fl} \models Compl(Q_{fb})$ by Theorem 3.3. This implies that $\bar{C} \models_{fed} Compl(Q_{fb})$ from Proposition 4.9.

Using the above proposition, we can check the completeness of a basic query in the federated setting with the criterion in Theorem 3.3, as shown in the above example.

We now introduce *smart rewritings* that allow us to evaluate each triple pattern of a query over a single source such that the join of the results is the same as the result of the original query executed over the union of all sources. Let us first define that a federated query \bar{Q} is a *federated rewriting* of a basic query Q if $\bar{Q}^{fl} = Q$. In other words, by dropping the SERVICE calls from \bar{Q} , we obtain Q .

Definition 4.11 (*Smart Rewriting*). Let \bar{C} be a set of indexed completeness statements, Q a basic query, and \bar{Q} a federated rewriting of Q . Then, \bar{Q} is a *smart rewriting* of Q with regard to \bar{C} if $\llbracket \bar{Q} \rrbracket_{(j_0, \bar{G}^a)} = \llbracket Q \rrbracket_{\cup_{j \in J} G^a}$ for every (\bar{G}^a, G^i) satisfying \bar{C} and every $j_0 \in J$.

This definition emphasizes that indexed completeness statements not only express the relationship between an available graph and the ideal graph but also the relationship among available graphs themselves: two different available graphs with the same completeness statement have exactly the same (complete) data that is captured by the statement. As an illustration, consider Example 4.5, in which both DBpedia and LMDb have the same completeness statement C_{dir} about movies directed by Tarantino. Therefore, the set of Tarantino movies in DBpedia must be identical to that in LMDb. In the next section, we describe sufficient and necessary conditions for the existence of such smart rewritings and ways for finding them.

4.2 Finding Smart Rewritings

Now that we have the notion of smart rewritings, the question arises as to how to find one. Let us first define an *indexed triple* (t, j) as a pair of a triple t and an IRI j . Next, we define a *federated transfer operator* to generate, given a graph, a set of indexed triples with regard to a set of indexed completeness statements.

Definition 4.12. Let \bar{C} be a set of indexed completeness statements and G a graph. We define the operator $T_{\bar{C}}$ over G as follows:

$$T_{\bar{C}}(G) = \bigcup_{(c,k) \in \bar{C}} \{(t, k) \mid t \in \llbracket Q_C \rrbracket_G\}.$$

The operator $T_{\bar{C}}$ is similar to the transfer operator T_C in Equation (3), except that it now labels the resulting triples with the indexes of the indexed completeness statements that can derive them from G . Note that a set of indexed triples can be flattened. For a set \bar{T} of indexed triples, we define the flattening as $\bar{T}^{fl} = \{t \mid (t, k) \in \bar{T}\}$. We now present a proposition that provides a syntactic characterization of the completeness of a basic query with regard to a set of indexed completeness statements.

PROPOSITION 4.13. Let \bar{C} be a set of indexed completeness statements and $Q = (W, P)$ be a basic query. Then,

$$\bar{C} \models_{fed} Compl(Q) \quad \text{iff} \quad \tilde{P} = (T_{\bar{C}}(\tilde{P}))^{fl}.$$

According to this proposition, we can verify whether \bar{C} entails the completeness of a basic query Q by checking if we obtain the prototypical graph \tilde{P} after applying the federated transfer operator $T_{\bar{C}}$ to \tilde{P} and then flattening the result. Let us now revisit our running example.

Example 4.14. Consider the set \bar{C} of indexed completeness statements and the query Q_{fb} from above. We have the prototypical graph

$$\tilde{P}_{fb} = \{ (\tilde{m}, a, Movie), (\tilde{m}, director, tarantino), (\tilde{m}, likes, \tilde{l}) \}.$$

The result of the federated transfer operator $T_{\bar{C}}(\tilde{P}_{fb})$ is the set of indexed triples

$$\{ ((\tilde{m}, a, Movie), DBPe), ((\tilde{m}, director, tarantino), DBPe), ((\tilde{m}, a, Movie), LMDBe), \\ ((\tilde{m}, director, tarantino), LMDBe), ((\tilde{m}, likes, \tilde{l}), FBe) \}.$$

It is easy to see that $(T_{\bar{C}}(\tilde{P}_{fb}))^{fl} = \tilde{P}_{fb}$. Therefore, by Proposition 4.13, it is the case that $\bar{C} \models_{fed} Compl(Q_{fb})$.

According to the above proposition, if $Q = (W, P)$ is complete with regard to \bar{C} , then for every triple t_i occurring in $\{ t_1, \dots, t_n \} = \tilde{P}$, we can find a corresponding triple $(t_i, k_i) \in T_{\bar{C}}(\tilde{P})$. Therefore, we can choose a set of indexed triples $\tilde{S} = \{ (t_1, k_1), \dots, (t_n, k_n) \} \subseteq T_{\bar{C}}(\tilde{P})$, such that it satisfies two properties: (i) *uniqueness*: for each pair of indexed triples (t_i, k) , $(t_i, k') \in \tilde{S}$, it is the case that $k = k'$, that is, the index of t_i is unique; and (ii) *covering*: it is the case that $\tilde{S}^{fl} = \tilde{P}$. Hence, each triple of \tilde{P} occurs exactly once in \tilde{S} . We call such a set of indexed triples a *smart set*. It immediately follows that such smart sets always exist whenever $\bar{C} \models_{fed} Compl(Q)$. We next define a transformation from a smart set to a graph pattern with the SERVICE operator.

Definition 4.15 (SERVICE Transformation). Let $\tilde{T} = \{ (t_1, k_1), \dots, (t_m, k_m) \}$ be a smart set with regard to a set \bar{C} of indexed completeness statements and a query Q . For each indexed triple (t_i, k_i) in \tilde{T} , we create a SERVICE pattern (SERVICE k_i ($\tilde{id}^{-1}(t_i)$)), where \tilde{id}^{-1} is the inverse of the freeze mapping \tilde{id} . We then define the transformation from \tilde{T} to a graph pattern with the SERVICE operator as follows:

$$P_{\tilde{T}} = (\text{SERVICE } k_1 (\tilde{id}^{-1}(t_1))) \text{ AND } \dots \text{ AND } (\text{SERVICE } k_m (\tilde{id}^{-1}(t_m))).$$

As an alternative, one can also put together all triple patterns that go into the same endpoint. Let us give an example of a SERVICE transformation.

Example 4.16. Consider again the set $T_{\bar{C}}(\tilde{P}_{fb})$ of indexed triples. The following is a smart set (i.e., satisfying the uniqueness and covering properties) of that set:

$$\{ ((\tilde{m}, a, Movie), LMDBe), ((\tilde{m}, director, tarantino), LMDBe), ((\tilde{m}, likes, \tilde{l}), FBe) \}.$$

The SERVICE transformation of that smart set is exactly the body of the federated query \bar{Q}_{fb} .

The next lemma basically says, for a basic query that can be answered completely with regard to a set of indexed completeness statements, we can always find a smart rewriting of the basic query and such a smart rewriting is complete with regard to the set of indexed completeness statements.

LEMMA 4.17. Let \bar{C} be a set of indexed completeness statements and $Q = (W, P)$ be a basic query such that $\bar{C} \models_{fed} Compl(Q)$. Moreover, let \tilde{S} be a smart set with regard to \bar{C} and Q , and $P_{\tilde{S}}$ be the SERVICE transformation from the smart set \tilde{S} . Then,

$$\bar{Q} = (W, P_{\tilde{S}}) \text{ is a smart rewriting of } Q \text{ with regard to } \bar{C} \text{ such that } \bar{C} \models_{fed} Compl(\bar{Q}).$$

It also holds that a basic query Q can only have a smart rewriting with regard to a set \bar{C} of indexed completeness statements if $\bar{C} \models_{fed} Compl(Q)$.

THEOREM 4.18. *Let \bar{C} be a set of indexed completeness statements and let $Q = (W, P)$ be a basic query. Then,*

$$\text{there exists a smart rewriting of } Q \text{ with regard to } \bar{C} \quad \text{iff} \quad \bar{C} \models_{fed} Compl(Q).$$

The following example is a consequence of this theorem.

Example 4.19. Consider again the set \bar{C} of indexed completeness statements and the query Q_{fb} , as above. From Example 4.14, we have that $\bar{C} \models_{fed} Compl(Q_{fb})$. On the other hand, from the smart set in Example 4.16, we can construct a smart rewriting of Q_{fb} with regard to \bar{C} , which is the federated query \bar{Q}_{fb} . Note that single data source approaches cannot guarantee the completeness of Q_{fb} , as we need to join query answers from more than one data source.

We have shown that for a basic query that is complete with regard to a set of indexed completeness statements, rather than evaluating the query over the union of all available sources, we evaluate a smart rewriting of the query, which always exists. This way, we send the various parts of the query only to those individual sources that can provide complete results for these parts and join the results. Eventually, we retrieve the complete query answer in this way.

Practical Considerations about Smart Rewriting. In this work, we consider the problem of checking query completeness in a crisp manner: either completeness can be guaranteed or not. In the federated case, Theorem 4.18 states that only a complete query can be rewritten in a smart way. Nevertheless, in the setting in which only a subset of the query can be found to be complete, one possible way to still efficiently answer the query over federated data sources is as follows: for those parts of the query that can be guaranteed to be complete, they can be evaluated exclusively over a data source that is guaranteed to be complete by the completeness statements. To obtain a complete answer for remaining parts, we send them to every data source.

Another practical consideration is where to store the completeness statements. One possible way is to have a completeness hub, which stores completeness statements of multiple data sources and provides federated completeness checking. A system to demonstrate such a functionality has been proposed by Darari et al. (2014). A decentralized approach can be taken in the following way: Since the query processor can query sources for data, it is conceivable to query data sources for their completeness statements. To exclude irrelevant statements, the query processor could ask the sources for statements whose constants are a subset of the constants of the query. Using the terminology of Section 6, the processor would apply the *constant-relevance principle*, which is underlying the optimization techniques that we present later and that turned out to be effective in significantly reducing the number of statements to consider in our experiments.

Schema Heterogeneity. In practice, we observed three different scenarios of schema usages. The easiest is when different data sources use the same schema. In this case, completeness statements can readily be employed. The second case is when data from different sources is expressed with different schemas. This is one of the motivations for the RDFS incorporation (which supports schema mapping via `rdfs:subClassOf` and `rdfs:subPropertyOf`) to our completeness framework, as in Section 3.4. Example 3.12 has shown that, despite using a different term (i.e., `Film`), the query can still be guaranteed to be complete by the completeness statement of all movies, thanks to the RDFS mapping between `Film` and `Movie`. Moreover, the increased attention to global schemas (e.g., `schema.org`) will also mitigate this issue. For example, the DBpedia schema is already mapped to

schema.org.¹³ Parts of Wikidata (Erxleben et al. 2014), one of the fastest growing KBs with RDF support, have also been mapped to schema.org with at least 464 class and property mappings.¹⁴ In this case, one can create completeness statements about DBpedia and Wikidata using terms from schema.org. Given the centrality of DBpedia and Wikidata in terms of their linkedness to many other data sources on the Web of Linked Data, this can be an advantageous situation for interoperable completeness statements over multiple data sources. Additionally, there is increased agreement on common vocabularies for modeling certain entity types on the Web of Linked Data, as exemplified by the increase usage of the FOAF vocabulary from 27% of all datasets in 2011 to 69% in 2014, and the Dublin Core vocabulary from 31% in 2011 to 56% in 2014 (Schmachtenberg et al. 2014). In fact, completeness statements produced using those shared vocabularies can be compatible across different sources, with the following characteristic: the more widely vocabularies are used, the better for the interoperability of completeness statements.

The last scenario is the most difficult one: different data sources employ not only different schemas but also different structures. For example, one data source might represent movies directed by Tarantino using two triples, such as (*reservoirDogs*, *a*, *Movie*), (*reservoirDogs*, *directedBy*, *Tarantino*), instead of one triple, such as (*reservoirDogs*, *a*, *MovieDirectedByTarantino*). In this situation, RDFS reasoning is not sufficient to map between those two different representations. More complicated rule-based mappings might provide a solution, yet it is beyond the scope of this work. For more details on tools and approaches to schema mapping and data integration, we refer the reader to Doan et al. (2012), Euzenat and Shvaiko (2013), and Heath and Bizer (2011).

5 COMPLETENESS REASONING WITH TIME

When creating completeness statements about a data source, one might make the assumption that the data source, regardless of time, is always complete for parts of data captured by the statements. This is true under the following circumstances: the data by nature will not change anymore (e.g., all movies starring Charlie Chaplin and all actors in *Reservoir Dogs*) or, if the data may still change, the data source enforces a synchronization mechanism to immediately capture new facts in the real world. Nevertheless, there might be situations in which such a synchronization is unlikely, say, when the data provider is not an authority or the data originates from crowdsourcing. Consequently, completeness statements can become out of date, that is, the data in the source captured by the statements does not reflect the newly updated complete facts in the real world. In this section, we discuss how to extend completeness statements to cope with data dynamicity over time and reason about query completeness given such extended statements.

5.1 Motivating Example

To deal with data dynamicity, a time extension to completeness statements is a necessity. While several works in temporal RDF existed with respect to representation and reasoning (Gutiérrez et al. 2005; Lopes et al. 2010) and indexing (Pugliese et al. 2008; Tappolet and Bernstein 2009), none took into account the data completeness aspect.¹⁵ In this work, by dynamicity we refer to any addition of data, that is, new information is added without invalidating old information. In other words, we work with graphs whose information is *monotonically nondecreasing*. Many domains of information typically follow this characteristic, for instance, publications of a researcher, movies of an actor, and children of a person. Consider the statement “Crew of Tarantino movies are complete” over a data source. Given the fact that Tarantino is currently an active director, the data captured by

¹³<http://wiki.dbpedia.org/services-resources/ontology>.

¹⁴As of December 5, 2017, by the Wikidata SPARQL query: <http://bit.ly/wikidataToSchemaOrgMappings>.

¹⁵Also in our work, data is left as is with no explicit time annotation. Only completeness statements have time annotations.

the statement is likely to grow. However, suppose that the data source fails to capture an update of the data. What happens then is that the completeness statement over the source provides a false claim. On the other hand, consider the statement “Crew of Tarantino movies up to 2012 are complete,” which is the statement as before, now with a date. The date represents the temporal scope of the statement, giving a boundary up to when it is complete, that is, up to 2012. Thus, the statement is still correct, even if there are new Tarantino movies released after 2012 whose crew are not captured by the data source. We call such a statement a *bounded completeness statement*.

Now, consider the statement “Movies starring Chaplin are complete and there will not be any updates.” This statement is plausible since Chaplin passed away in 1977. A data source with the statement is therefore always complete for movies starring Chaplin regardless of time since the data can no longer grow. We call such a statement an *unbounded completeness statement*.

Reasoning about query completeness based on statements with a time extension must be approached differently. For this reason, we introduce the notion of the *guaranteed completeness date* of a query, that is, the latest date on which complete query results are guaranteed to be contained in the actual query results.

Consider again the statement “Crew of Tarantino movies up to 2012 are complete.” Suppose that we have another statement, “Cast of Tarantino movies up to 2017 are complete.” If we query for people who are both cast and crew of Tarantino movies, we can be certain that the query answers will be complete up to 2012, since the crew of Tarantino movies are complete up to that time and even further for the cast. However, from 2013 onwards, the query completeness cannot be guaranteed, as we might be missing some crew members of Tarantino movies released after 2012. We therefore call 2012 the *guaranteed completeness date* of the query.

In contrast, let us consider again the statement “Movies starring Chaplin are complete and there will not be any updates.” If we are now querying for movies starring Chaplin, the results of this query will be complete and will be so for query results at any time in the future. Therefore, the *guaranteed completeness date* of the query is infinity.

Not all queries have a *guaranteed completeness date*, depending on the statements we have. Consider again the statement “Cast of Tarantino movies up to 2017 are complete” and consider the query asking for all spouses of the cast of Tarantino movies. Since we do not have any completeness assertion about the spouses, the completeness of that query cannot be guaranteed with regard to any date; thus, there is no *guaranteed completeness date* for the query.

5.2 Time-Extended Completeness Framework

We now formalize the extended completeness framework and its semantics. We define a *date* as an element $d \in \mathbb{N} \cup \{\infty\}$.¹⁶ We use natural numbers, as we can reduce dates of various granularities (e.g., years, seconds, and calendar dates) to them. We assume a fixed constant $now \in \mathbb{N}$.

Timestamped Completeness Statements. The first step is to incorporate timestamps in completeness statements.

Definition 5.1 (Timestamped Completeness Statement). A *timestamped completeness statement* is of the form

$$\hat{C} = Compl(P_1 | P_2, d),$$

where $Compl(P_1 | P_2)$ is a completeness statement as seen before, now extended with a date d , such that either $d \leq now$ or $d = \infty$. In the first case, we say that \hat{C} is *bounded*, whereas in the second case, \hat{C} is *unbounded*.

¹⁶Without loss of generality, our framework also supports continuous domains (e.g., \mathbb{R}), given that the discretization of all known timestamps gives a discrete space again.

Example 5.2. Consider the statements “Crew of Tarantino movies up to 2012 are complete,” “Cast of Tarantino movies up to 2017 are complete,” and “Movies starring Chaplin are complete and there will not be any updates,” as above. They can be represented formally as

$$\begin{aligned}\hat{C}_{crew} &= \text{Compl}(\{ (?m, crew, ?c), (?m, a, TarantinoMov) \} \mid \emptyset, 2012) \\ \hat{C}_{cast} &= \text{Compl}(\{ (?m, cast, ?c), (?m, a, TarantinoMov) \} \mid \emptyset, 2017) \\ \hat{C}_{chap} &= \text{Compl}(\{ (?m, a, ChaplinMov) \} \mid \emptyset, \infty).\end{aligned}$$

Over a set of timestamped completeness statements, we define the operator *date* to extract the dates of the statements. To select timestamped completeness statements based on their dates, over a set \hat{C} of timestamped completeness statements and a date d , the selection $\hat{C}_{\geq d}$ is defined as

$$\hat{C}_{\geq d} = \{ \hat{C} \in \hat{C} \mid \text{date}(\hat{C}) \geq d \}.$$

The selection $\hat{C}_{=d}$ is defined analogously. As before, we associate to a statement \hat{C} , a CONSTRUCT query $Q_{\hat{C}} = (\text{CONSTRUCT } P_1 \ P_1 \cup P_2)$. Over a graph G , the transfer operator $T_{\hat{C}}(G)$ is defined similarly to that in Equation (3) in Section 2.1, in which we take the union of the results of the evaluation $\llbracket Q_{\hat{C}} \rrbracket_G$ of all $\hat{C} \in \hat{C}$.

RDF Representation of Timestamped Completeness Statements. To represent timestamped completeness statements in RDF, we propose using the datatype representation from the XML Schema Definition (XSD) namespace to represent noninfinity dates, which can also be of various granularities, such as years and calendar dates.¹⁷ To represent infinity, we introduce in our vocabulary¹⁸ the term *infinity*. Then, we create the property name *hasTimestamp* to link between completeness statements and their timestamps.

Incomplete Data Series. The models of timestamped completeness statements are incomplete data series. An *incomplete data series* (or, for short, a series) \mathcal{S} is a pair of an available graph and a sequence of ideal graphs, of the form

$$\mathcal{S} = \left(G_{now}^a, \left(G_1^i, G_2^i, \dots, G_{now}^i, \dots \right) \right),$$

such that (G_{now}^a, G_{now}^i) is an incomplete data source and it holds that $G_d^i \subseteq G_{d+1}^i$ for all pairs G_d^i, G_{d+1}^i of ideal graphs. We have one available graph only since we focus on the actual available graph, that is, the state of the available graph we have now. On the other hand, we have a sequence of ideal graphs to represent data dynamicity over time with regard to the real world.

Example 5.3 (Incomplete Data Series). Let *now* = 2017 and

$$\mathcal{S}_{mov} = \left(G_{now}^a, \left(G_1^i, \dots, G_{2012}^i, \dots, G_{now}^i, \dots \right) \right)$$

be a series about Tarantino and Chaplin movies that can be graphically represented as in Figure 6.¹⁹ Note that, in this example, the set of movies starring Chaplin will no longer grow (i.e., only *The Kid*) and any other ideal graph G_k^i not shown in the figure is defined accordingly.

We now formalize when a series satisfies a timestamped completeness statement. A series \mathcal{S} satisfies a bounded timestamped completeness statement $\hat{C} = \text{Compl}(P_1 \mid P_2, d)$, written as $\mathcal{S} \models \hat{C}$, if all the triples constructed by evaluating $Q_{\hat{C}}$ over the ideal graph at date d are in the actual available graph, formalized as $\llbracket Q_{\hat{C}} \rrbracket_{G_d^i} \subseteq G_{now}^a$. Note that this implies that $\llbracket Q_{\hat{C}} \rrbracket_{G_{d'}^i} \subseteq G_{now}^a$ for all $d' \leq d$ by the definition of a series. If the statement is unbounded, then the comparison for

¹⁷<http://www.w3.org/2001/XMLSchema>.

¹⁸<http://completeness.inf.unibz.it/ns>.

¹⁹For the sake of example, we only use toy data.

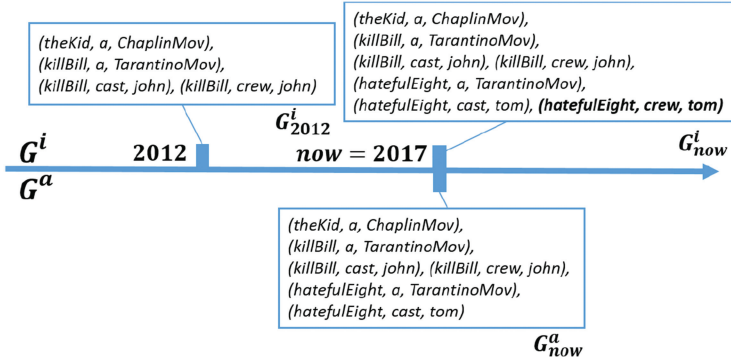


Fig. 6. An incomplete data series about Tarantino and Chaplin movies.

completeness is made over all ideal graphs: for all $d \in \mathbb{N}$, it must hold that $\mathcal{S} \models \text{Compl}(P_1 \mid P_2, d)$. Given a set $\hat{\mathcal{C}}$ of timestamped completeness statements and a series \mathcal{S} , we define that $\mathcal{S} \models \hat{\mathcal{C}}$, if for all $\hat{C} \in \hat{\mathcal{C}}$, it holds that $\mathcal{S} \models \hat{C}$.

Example 5.4. Consider the series \mathcal{S}_{mov} and the statements \hat{C}_{crew} , \hat{C}_{cast} , and \hat{C}_{chap} , as above. Then, it holds that $\mathcal{S}_{mov} \models \hat{C}_{crew}$ because the result of the query evaluation

$$\llbracket Q_{\hat{C}_{crew}} \rrbracket_{G_{2012}^i} = \{ (killBill, crew, john), (killBill, a, TarantinoMov) \}$$

is contained in G_{now}^a . For a similar reason, $\mathcal{S}_{mov} \models \hat{C}_{cast}$ also holds. Moreover, it is the case that $\mathcal{S}_{mov} \models \hat{C}_{chap}$ since for G_{2012}^i , G_{2017}^i , and any other ideal graph G_k^i in \mathcal{S}_{mov} , the result (i.e., the graph $\{ (theKid, a, ChaplinMov) \}$) of the query $Q_{\hat{C}_{chap}}$ evaluated over them is contained in G_{now}^a .

Query Completeness at a Date. To describe query completeness at date d , we use $\text{Compl}(Q, d)$. A series \mathcal{S} satisfies $\text{Compl}(Q, d)$ with $d \in \mathbb{N}$, written as $\mathcal{S} \models \text{Compl}(Q, d)$, if evaluating Q over the ideal graph at d gives results that are all contained in the results of evaluating Q over the actual available graph, formalized as $\llbracket Q \rrbracket_{G_d^i} \subseteq \llbracket Q \rrbracket_{G_{now}^a}$. Furthermore, a series \mathcal{S} satisfies the unbounded version of query completeness, written as $\mathcal{S} \models \text{Compl}(Q, \infty)$, if for all $d \in \mathbb{N}$, it holds that $\mathcal{S} \models \text{Compl}(Q, d)$.

Example 5.5. To say that the query asking for all people who are both cast and crew of Tarantino movies up to 2012 is complete, we can use $\text{Compl}(Q_{cc}, 2012)$ where

$$Q_{cc} = (\{ ?m, ?c \}, \{ (?m, cast, ?c), (?m, crew, ?c), (?m, a, TarantinoMov) \}).$$

As we can see, $\llbracket Q_{cc} \rrbracket_{G_{2012}^i}$ returns $(?m \mapsto killBill, ?c \mapsto john)$ and is contained in $\llbracket Q_{cc} \rrbracket_{G_{now}^a}$, therefore $\mathcal{S}_{mov} \models \text{Compl}(Q_{cc}, 2012)$. In contrast, $\llbracket Q_{cc} \rrbracket_{G_{2017}^i}$ also returns $(?m \mapsto hatefulEight, ?c \mapsto tom)$, which is not in $\llbracket Q_{cc} \rrbracket_{G_{now}^a}$, therefore $\mathcal{S}_{mov} \not\models \text{Compl}(Q_{cc}, 2017)$.

Having defined timestamped completeness statements and query completeness at a date, the question arises regarding how to actually check the entailment between them. Given a set $\hat{\mathcal{C}}$ of timestamped completeness statements, a query Q , and a date d , we say that $\hat{\mathcal{C}}$ entails query completeness at d , written as $\hat{\mathcal{C}} \models \text{Compl}(Q, d)$, if for all $\mathcal{S} \models \hat{\mathcal{C}}$, it is the case that $\mathcal{S} \models \text{Compl}(Q, d)$. The following lemma gives us a syntactic characterization to decide whether $\hat{\mathcal{C}} \models \text{Compl}(Q, d)$. It says that the query completeness at d is entailed by $\hat{\mathcal{C}}$ iff the prototypical graph \tilde{P} of Q is contained in the result of the transfer operator applied to \tilde{P} , using only the statements in $\hat{\mathcal{C}}$ with dates $\geq d$.

LEMMA 5.6 (ENTAILMENT OF QUERY COMPLETENESS AT A DATE). *Let \hat{C} be a set of timestamped completeness statements, let $Q = (W, P)$ be a query, and let d be a date. Then,*

$$\hat{C} \models \text{Compl}(Q, d) \quad \text{iff} \quad \tilde{P} \subseteq T_{\hat{C}_{\geq d}}(\tilde{P}).$$

Guaranteed Completeness Date. We now formalize the notion of guaranteed completeness date, introduced in the preceding examples. The *guaranteed completeness date* of a query Q with regard to a set \hat{C} of timestamped completeness statements is the latest date d such that the entailment $\hat{C} \models \text{Compl}(Q, d)$ holds, formally $\text{gcd}(Q, \hat{C}) = \max\{d \in \mathbb{N} \cup \{\infty\} \mid \hat{C} \models \text{Compl}(Q, d)\}$. We also define $\max\{\} = -\infty$, and note that cases in which $\text{gcd}(Q, \hat{C}) = -\infty$ correspond to the query Q not having any completeness date.

Example 5.7. Consider the set of statements $\hat{C} = \{\hat{C}_{crew}, \hat{C}_{cast}, \hat{C}_{chap}\}$ and the query Q_{cc} , as above. It is the case that $\text{gcd}(Q_{cc}, \hat{C}) = 2012$ for the following reasons. While the statement \hat{C}_{chap} obviously does not contribute at all to the guaranteed completeness date of the query, the statements \hat{C}_{crew} and \hat{C}_{cast} do contribute. If we execute the query, we can be complete up to 2012, since the crew of Tarantino movies are complete up to that time, as guaranteed by \hat{C}_{crew} , and even further for the cast, as guaranteed by \hat{C}_{cast} . From 2013 onward, however, the query completeness cannot be guaranteed, as some crew members of Tarantino movies might be missing. Therefore, 2012 is the guaranteed completeness date.

5.3 Computing the Guaranteed Completeness Date

We now analyze how the guaranteed completeness date of a query can be computed. By Lemma 5.6, we can replace the entailment $\hat{C} \models \text{Compl}(Q, d)$ in the definition of the guaranteed completeness date by its syntactic characterization $\tilde{P} \subseteq T_{\hat{C}_{\geq d}}(\tilde{P})$. In this way, we compute the maximum date from all the dates d in \hat{C} such that query completeness can be guaranteed by using only the statements having a date $d' \geq d$, as shown in the following theorem.

THEOREM 5.8 (COMPUTING THE GUARANTEED COMPLETENESS DATE). *Let $Q = (W, P)$ be a query and let \hat{C} be a set of timestamped completeness statements. Then,*

$$\text{gcd}(Q, \hat{C}) = \max \left\{ d \in \text{date}(\hat{C}) \mid \tilde{P} \subseteq T_{\hat{C}_{\geq d}}(\tilde{P}) \right\}.$$

In the following example, we apply the above theorem to compute the guaranteed completeness date of our running example.

Example 5.9. Consider the statements $\hat{C} = \{\hat{C}_{crew}, \hat{C}_{cast}, \hat{C}_{chap}\}$ and the query $Q_{cc} = (W_{cc}, P_{cc})$, as above. The set of the dates is $\text{date}(\hat{C}) = \{2012, 2017, \infty\}$. Then, we have that

- $\tilde{P}_{cc} \subseteq \{(\tilde{m}, \text{cast}, \tilde{c}), (\tilde{m}, \text{crew}, \tilde{c}), (\tilde{m}, a, \text{TarantinoMov})\} = T_{\hat{C}_{\geq 2012}}(\tilde{P}_{cc})$
- $\tilde{P}_{cc} \not\subseteq \{(\tilde{m}, \text{cast}, \tilde{c}), (\tilde{m}, a, \text{TarantinoMov})\} = T_{\hat{C}_{\geq 2017}}(\tilde{P}_{cc})$

Thus, we can conclude that $\text{gcd}(Q_{cc}, \hat{C}) = 2012$.

From the theorem above, we observe the following computational complexity of the decision version of computing the guaranteed completeness date. It shows that adding a time extension does not increase the complexity of completeness reasoning as it is still in NP.

COROLLARY 5.10 (COMPLEXITY OF DECIDING THE GUARANTEED COMPLETENESS DATE). *Deciding whether $\text{gcd}(Q, \hat{C}) \geq d$, given a query Q , a set \hat{C} of timestamped completeness statements, and a date d , is NP-complete.*

ALGORITHM 1: FINDGCD

Input: The prototypical graph \tilde{P} of a query, a set of timestamped completeness statements \hat{C}
Output: The guaranteed completeness date d
 $P' \leftarrow \emptyset$
 $D \leftarrow \text{date}(\hat{C}) \cup \{-\infty\}$
while $\tilde{P} \not\subseteq P'$ and $D \neq \emptyset$ **do**
 $d \leftarrow \text{extractMax}(D)$
 $P' \leftarrow P' \cup T_{\hat{C}=d}(\tilde{P})$
end
return d

Fig. 7. Algorithm for finding the guaranteed completeness date.

Algorithm for Finding the Guaranteed Completeness Date. Based on Theorem 5.8, a naïve way to compute the guaranteed completeness date is, for every $d \in \text{date}(\hat{C})$, to repeatedly compute $T_{\hat{C} \geq d}(\tilde{P})$ and then take the maximum of the dates d such that $\tilde{P} \subseteq T_{\hat{C} \geq d}(\tilde{P})$. This has a drawback since we may be reevaluating the CONSTRUCT query of a statement over \tilde{P} , even if the result is always the same. We could improve the computation by using binary search, as $\text{date}(\hat{C})$ has a natural order and $T_{\hat{C} \geq d}(\tilde{P})$ is monotonic in d . As a consequence, checking $\tilde{P} \subseteq T_{\hat{C} \geq d}(\tilde{P})$ would be done only $\log(|\text{date}(\hat{C})|)$ times instead of $|\text{date}(\hat{C})|$ times.

Now, we observe the following. For a date d , the result of $T_{\hat{C} \geq d}(\tilde{P})$ is, in fact, the union of all $T_{\hat{C}=d'}(\tilde{P})$, where $d \leq d' \leq \max(\text{date}(\hat{C}))$. Thus, we can compute $T_{\hat{C} \geq d}(\tilde{P})$ in an incremental way from the latest d' . An algorithm to find the guaranteed completeness date would incrementally compute the union from the latest date in $\text{date}(\hat{C})$ to the earliest date in $\text{date}(\hat{C})$ while on the way checking if \tilde{P} is already included. If that is the case, we can just stop and return the current date in the iteration as the guaranteed completeness date. In this way, each corresponding CONSTRUCT query of a timestamped completeness statement needs to be executed at most only once over \tilde{P} .

We formalize this as the algorithm FINDGCD in Figure 7. The algorithm takes as input the prototypical graph \tilde{P} of Q and a set of timestamped completeness statements \hat{C} . At first, we assign the empty set to P' , which will store the application results of the transfer operator $T_{\hat{C}=d}(\tilde{P})$, and assign all the dates in \hat{C} and $-\infty$ to D . We then perform a while loop with the conditions “ $\tilde{P} \not\subseteq P'$ ” to check that \tilde{P} has not been included in the accumulation, and “ $D \neq \emptyset$ ” to ensure that we still have some dates in D . For every loop, we execute $\text{extractMax}(D)$ to return the latest date d in D and remove it from D . Next, we add to P' the result of $T_{\hat{C}=d}(\tilde{P})$. At the end of the algorithm, we will return d , which is the guaranteed completeness date of Q with regard to \hat{C} . Note that when $d = -\infty$, the transfer operator $T_{\hat{C}=-\infty}(\tilde{P})$ would return an empty set, since $\hat{C}_{=-\infty} = \emptyset$ by definition.

Using this incremental computation, completeness checking requires evaluation of each completeness statement at most once, which means that completeness checking with time is no more complex than completeness checking without time.

Nonmonotonic Data Dynamicity. In this work, we assume that graphs are monotonically growing. In other words, the information in those graphs cannot shrink but rather will only grow over time. As such, a limitation of our work is that it applies only to invariable facts, that is, facts that hold eternally. If facts can also become invalid, completeness statements may then only capture the completeness of facts “at a specific point of time,” as opposed to “up to a specific point of time.” This can be problematic for checking the completeness of queries that require joins. A possible

solution is to also incorporate time into RDF graphs. As an illustration, suppose that facts about people being students have a temporal qualification. Then, we can say that we are complete for all students of Bolzano University until 2016 in the sense that we have a complete (timestamped) record of people who were Bolzano students up to 2016. This would make little sense for triples without timestamps since—say, after graduation—people are no longer a student (and hence, the facts have to be removed).

In this section, we have motivated, formalized, and developed a technique for completeness reasoning with time over basic queries. This serves also as the basis for time-aware completeness reasoning under the RDFS entailment regime and the federated setting, which can be realized with relatively minimal effort. An important question now is how in practice we may perform completeness reasoning at all. In the next section, we will see how completeness reasoning can deal with large sets of completeness statements.

6 EXPERIMENTAL EVALUATION

We now experimentally evaluate completeness reasoning. We first focus on completeness reasoning over basic queries and then extend it to the RDFS semantics. Before delving further into the discussion of the evaluation, we shall introduce a *relevance principle*, which allows reduction of the number of completeness statements considered in the reasoning. Later, we will show that this principle can improve the feasibility of completeness reasoning via experimental evaluations with real query logs from DBpedia, Linked Geo Data, and Semantic Web Dog Food SPARQL endpoints.

6.1 Relevant Completeness Statements

Real-world RDF data sources may contain a large amount of data. This reflects in possibly large numbers of completeness statements to describe the completeness of data. At this point, the question about how fast completeness reasoning can be performed with a large amount of completeness statements naturally arises.

Let us first estimate the complexity of the completeness reasoning task, from which we will formulate a principle to optimize completeness reasoning. Let $Q = (W, P)$ be a query and let C be a set of completeness statements. According to Theorem 3.3, the task of completeness reasoning for basic queries is to check whether $T_C(\tilde{P}) = \tilde{P}$, where T_C is the transfer operator with regard to C and \tilde{P} is the prototypical graph of Q . While it is immediate to check the ‘ \subseteq ’ direction of the equality, the interesting part is the ‘ \supseteq ’ direction. This corresponds to finding, for each triple $(s, p, o) \in \tilde{P}$, a completeness statement $C \in C$ such that $(s, p, o) \in \llbracket Q_C \rrbracket_{\tilde{P}}$ (recall that $T_C(\tilde{P}) = \bigcup_{C \in C} \llbracket Q_C \rrbracket_{\tilde{P}}$). Hence, we only find statements that *potentially* match such a triple (s, p, o) .

Let $Q = (W, P)$ be a query, let C be a set of completeness statements, and let $\max Ln(C)$ be the maximum length (i.e., the maximum number of triple patterns) of statements in C . Take any $C \in C$; to evaluate the query Q_C over \tilde{P} , it is necessary to (consistently) map the triple patterns of Q_C to triples in \tilde{P} . Note that there are at most $|\tilde{P}|^{|Q_C|}$ possible ways to map triple patterns to triples, where $|Q_C|$ and $|\tilde{P}|$ stand for the number of triple patterns and triples in Q_C and \tilde{P} , respectively. Therefore, applying this reasoning to each statement in C leads to an overall runtime of $\mathcal{O}(|C| |\tilde{P}|^{\max Ln(C)})$. Obviously, from this observation, completeness reasoning is data independent, that is, the size of the RDF graph to which completeness statements are given does not matter.

As is customary in the database theory when analyzing the data complexity of query evaluation, we are assuming that Q is given while the set of completeness statements varies. Moreover, since completeness statements are basically also queries, we assume the maximum length of completeness statements to be bound by a constant. Under these assumptions, the complexity of reasoning is a function of the size of the set of completeness statements. Using a *plain completeness reasoner*,

which evaluates the CONSTRUCT queries of *all* completeness statements, can potentially lead to slow performance. However, according to Theorem 3.3, which characterizes completeness entailment of basic queries, for a complete query with n triple patterns, there is a set of no more than n completeness statements that already entails the completeness of that query. From this observation, the problem of finding exactly those n completeness statements is crucial. Despite the fact that there is no obvious way to identify a priori such a set, as in the worst case all statements need to be checked, we establish a principle in the following that allows one to rule out a significant number of irrelevant statements.

Constant-Relevance Principle. Let us now introduce a relevance principle for completeness statements. Consider the query asking for “All movies directed by Tarantino” and the statement “All cantons of Switzerland.” Intuitively, one can see that the statement does not contribute to the completeness of the query; in other words, the statement is *irrelevant to the query*.

We shall now set the *constant-relevance principle* as a way to distinguish between irrelevant and relevant completeness statements. The principle states that a completeness statement C can contribute to entailing query completeness only if all constants (or terms, which consist of IRIs and literals) of the completeness statement occur also in the query Q , that is, $const(C) \subseteq const(Q)$. We say that a statement satisfying this principle is *constant relevant*. The following proposition shows that if a statement is not constant relevant, then it does not contribute to completeness reasoning.

PROPOSITION 6.1. *Let C be a completeness statement and let $Q = (W, P)$ be a query. If C is not constant-relevant with regard to Q , then $\llbracket Q_C \rrbracket_{\mathcal{P}} = \emptyset$.*

Proposition 6.1 opens up the problem of how to (efficiently) retrieve constant-relevant statements. It turns out that this problem is a variant of subset querying, which has been well studied (Helmer and Moerkotte 2003; Hoffmann and Koehler 1999; Savnik 2013). Our investigation on finding efficient techniques for completeness reasoning (Darari et al. 2016) showed that, despite its simplicity, standard hashing can be leveraged to perform subset querying in the context of completeness reasoning, outperforming the inverted indexing (Helmer and Moerkotte 2003) and tries (Hoffmann and Koehler 1999; Savnik 2013) techniques for most experiment cases. Thus, we concentrate our analysis on standard hashing. The idea of the standard hashing technique is to translate the problem of subset querying into one of evaluating exponentially many set equality queries. Hashing supports equality queries by performing retrieval of objects based on keys.

From a practical point of view, we store completeness statements according to their constant sets using a hash map. The maintenance cost (i.e., insert and delete operations) of the hash map is relatively low; hence, we focus more on the lookup operations. For each of the $2^{|const(Q)|} - 1$ nonempty subsets of $const(Q)$, we generate a set equality query using the hash map to retrieve the statements with exactly those constants and then compute the union of all the retrieved statements. For example, from the query $Q_{dir} = (\{ ?m \}, \{ (?m, a, Movie), (?m, director, tarantino) \})$, as in Example 2.7, the set of constants is $const(Q_{dir}) = \{ a, Movie, director, tarantino \}$, and the set equality queries are constructed from its nonempty subsets: $\{ a \}$, $\{ Movie \}$, \dots , and the set $const(Q_{dir})$ itself. Clearly, one possible limitation of this approach is the exponential blowup in the size of constants in queries. Nevertheless, in practice, queries contain a manageable number of constants, thus making this approach potentially suitable.

As discussed before in Section 3.4, completeness reasoning can also incorporate RDFS semantics. Then, RDFS-aware completeness checking, as characterized by Theorem 3.13, requires the closure computation before and after the T_C -application over the prototypical graph. In relation to the constant-relevance principle, completeness statements may then also capture inferred facts of the prototypical graph. Therefore, when RDFS schemas are considered, we need to extend the

relevance principle: a statement C is constant relevant to a query $Q = (W, P)$ with regard to a schema S if all constants in C are contained in the constants in $\tilde{id}^{-1}cl_S(\tilde{P})$, that is, the melting of the RDFS closure over the prototypical graph.

Our experimental evaluation was conducted with the aim to answer the following questions: (i) What is the overhead of completeness reasoning over querying? (ii) How do the two main completeness reasoning components, the hashmap lookup and the T_C -application, influence the overall completeness reasoning time? (iii) How do RDFS schemas influence completeness reasoning?

6.2 Experimental Setup

We created a framework for the experiments in Java using the Apache Jena library, an open-source Semantic Web library.²⁰ To implement completeness reasoning, we particularly relied on the ARQ module of Jena, which provides functionalities for SPARQL query processing. The retrieval of constant-relevant statements was implemented using a standard Java HashMap. We used the in-built RDFS reasoner of Jena via the method `ReasonerRegistry.getRDFSsimpleReasoner()` to compute RDFS closures.

The three ingredients that characterize our setting were queries, completeness statements, and RDFS schemas. *As for the queries*, we used openly available real query logs of DBpedia (DBP), Semantic Web Dog Food (SWDF), and Linked Geo Data (LGD), provided in the Linked SPARQL Queries (LSQ) dataset (Saleem et al. 2015). We extracted SELECT queries in the conjunctive fragment, which account for about 44% of the total number of SELECT queries, giving us around 467,000 queries in total.²¹ For example, one of the extracted queries from the DBpedia log is as follows:²²

$$Q_{cala} = (\{?abs\}, \{(Cala, abstract, ?abs), (Cala, a, MusicalArtist)\}).$$

As for the completeness statements, for each query $Q = (W, P)$ that we obtained above, we took the query's BGP body P and constructed one completeness statement for each element²³ of the powerset of the BGP P . For example, given the query Q_{cala} as above, the generated completeness statements are

- $Compl((Cala, abstract, ?abs) \mid \emptyset)$,
- $Compl((Cala, a, MusicalArtist) \mid \emptyset)$, and
- $Compl((Cala, abstract, ?abs), (Cala, a, MusicalArtist) \mid \emptyset)$.

This way of generating completeness statements creates the worst-case scenario in the sense that there are many relevant completeness statements for each query (as opposed to, for instance, just taking the exact BGP of the query as the pattern of the completeness statement).

Using query homomorphism techniques (Chandra and Merlin 1977), we removed duplicate completeness statements, that is, completeness statements whose CONSTRUCT query representations are equivalent to another query. Note that, here, we did not minimize the CONSTRUCT query representations of completeness statements. Instead, we checked if the statements could fully capture each other. For example, the following two completeness statements are equivalent:

- $Compl((?y, a, Person), (obama, spouse, ?y) \mid \emptyset)$, and
- $Compl((obama, spouse, ?spouse), (?spouse, a, Person) \mid \emptyset)$.

²⁰<http://jena.apache.org/>.

²¹As of July 1, 2017.

²²For the sake of example, we removed the prefixes and presented the abstract representation of the query.

²³Except the empty set.

Table 1. Overview of the Experiment Results

Endpoint	N_Q	N_C	$ Q $	t_{CR}	t_{QE}	N_S	t_{CRS}
SWDF	22,592	27,815	1.22	0.064 ms	8.3 ms	162	3.7 ms
DBP	336,139	406,589	1.13	0.068 ms	19.04 ms	6,977	285 ms
LGD	108,611	50,932	1.54	0.138 ms	36.2 ms	1,511	55 ms

Note: N_Q is the number of queries, N_C is the number of completeness statements, $|Q|$ is the average query length (i.e., number of triple patterns), t_{CR} is the average completeness reasoning time, t_{QE} is the average query evaluation time, N_S is the number of RDFS axioms, and t_{CRS} is the average RDFS-enabled completeness reasoning time.

In total, we generated about 485,000 unique completeness statements. Observe that, by construction, all queries are guaranteed to be complete.

As for the RDFSs, we took real-world schemas of those data sources: the Semantic Web Conference ontology,²⁴ the DBpedia ontology,²⁵ and the LGD ontology.²⁶ For each ontology, we extracted the RDFS axioms (i.e., `rdfs:domain`, `rdfs:range`, `rdfs:subClassOf`, and `rdfs:subPropertyOf`). The extracted schemas have various sizes (i.e., the number of RDFS axioms): 162 for SWDF, 6977 for DBP, and 1511 for LGD. The experiment framework (including the source code) is available online at <http://completeness.inf.unibz.it/completeness-experiment/>.

The goal of the evaluation was to measure completeness reasoning time for the queries. We distinguished between three cases of the experiments, depending on the endpoint of the queries: DBP, SWDF, or LGD. The experiments were run on a standard laptop in Windows 10 with an Intel Core i5 2.50GHz processor and 12GB RAM. Furthermore, for each query, we also took the query evaluation time, as provided by the LSQ dataset. The experiment machine for query evaluation had 16GB RAM and a 6-Core i7 3.40GHz CPU running Ubuntu 14.04.2 using Virtuoso 7.1 (Saleem et al. 2015). Note that the machine for query evaluation was relatively better than our machine for completeness reasoning.

6.3 Results and Discussion

Table 1 summarizes the results of the experiments. The number of queries varies greatly, with SWDF having the lowest and DBP having the highest. For the completeness statements, there are not many redundancies for DBP and SWDF as opposed to LGD. What is interesting is that mostly queries are short, close to one triple pattern, with a slight exception of LGD queries, whose average length is in the middle, between one and two triple patterns. On average, completeness reasoning time takes between 0.06 ms to 0.13 ms, which was very fast, thanks to the constant-relevance principle.

To compare with plain completeness reasoning (where *all* completeness statements are considered in reasoning), we took randomly 1000 queries for each case, and performed completeness reasoning, measuring, on average, 227 ms, 3,359 ms, and 434 ms, respectively. Thus, we have a considerable speed-up by using the constant-relevance principle, up to nearly 50,000 times faster. While for the plain reasoning the number of all completeness statements positively correlates with reasoning time, for the reasoning with the constant-relevance principle, this is not the case, as observed from the average reasoning time between DBP and LGD. With regard to query

²⁴http://data.semanticweb.org/ns/swc/swc_2009-05-09.html.

²⁵<http://wiki.dbpedia.org/services-resources/ontology>.

²⁶<http://downloads.linkedgeodata.org/releases/2014-09-09/>.

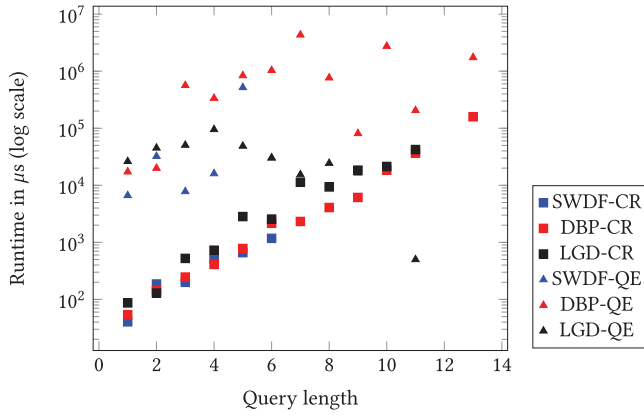


Fig. 8. Comparison of query length to completeness reasoning (CR) time and query evaluation (QE) time.

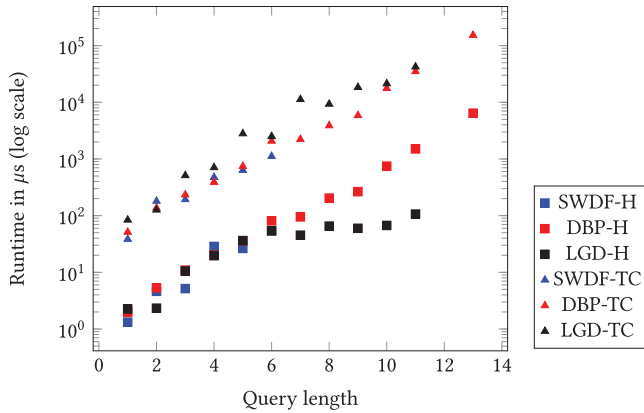


Fig. 9. Distribution of hashmap lookup time (H) and T_C -application time (TC) in completeness reasoning across different query lengths.

evaluation, completeness reasoning overall adds only a little overhead to query evaluation time: 0.5%, on average.

Figure 8 shows how overhead varies depending on query length. Note that the y-axis is in log scale. We can see that the data for query evaluation time shows no clear trend, whereas completeness reasoning time positively correlates with query length. Yet, in nearly all cases, query evaluation takes longer than completeness reasoning by several orders of magnitude. Note that, in all 3 query logs, most queries have short length, for instance, there are only fewer than 10 queries per length group for DBpedia queries with length greater than 6. Also, the worst case of completeness reasoning time in the figure is only 159 ms (for the DBP case in which the query length equals 13), which we consider reasonable.

Regarding the time for completeness reasoning with the constant-relevant principle, we can break this up into the time needed for the hashmap lookup for constant-relevant statements and the time for the T_C -application of those constant-relevant statements. Figure 9 shows how they distribute. As seen from the figure, the growth of the hashmap lookup time and T_C -application time are exponential in the query length. For the former, it is due to the exponential number of

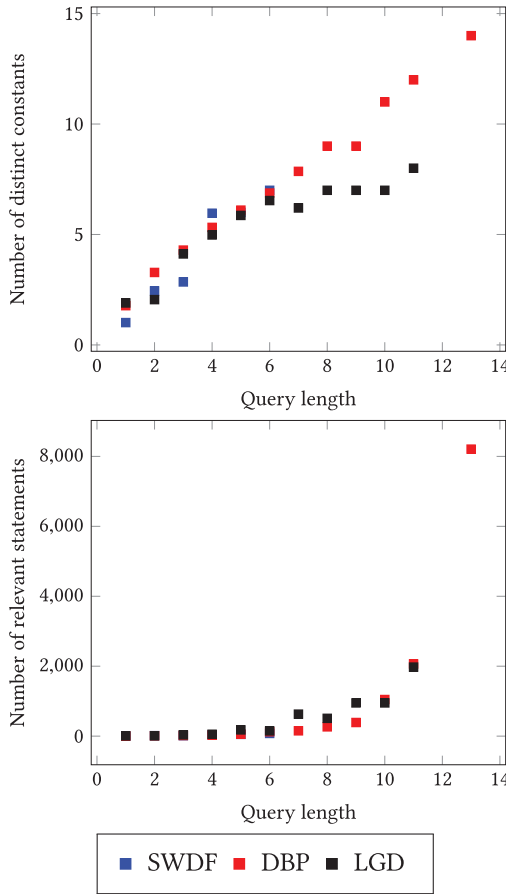


Fig. 10. Comparison of query length to number of distinct constants (top) and number of relevant statements (bottom).

set equality queries for retrieving constant-relevant statements. For the latter, it is due to how we generated completeness statements from the query, that is, we used the worst-case generation from the powerset of the query’s BGP. The longest hashmap lookup time is just 6 ms, whereas the longest T_C -application time is 152 ms, both of which we think are reasonably quick. Note that the hashmap lookup time relies on the number of constants, which, in our query logs, can be up to 14 constants. To see how many constants may break the hashmap lookup, we also performed an experiment with synthetic queries, in which we varied the number of constants from 1 to 32. From this experiment, we observed that hashmap lookup for up to 18 constants took slightly less than 100 ms. The time doubled until ultimately reaching 20 minutes with 32 constants.

Figure 10 (with a linear scale on the y-axis) provides an idea of how query length relates with the number of (distinct) constants in queries and the number of constant-relevant statements, respectively. In the top figure, it can be seen that the number of constants grows linearly with regard to query length, with a few exceptions. This is likely the reason for the exponential growth of hashmap lookup time in Figure 9, since the hashmap lookup depends exponentially on the number of constants (as per our analysis in Section 6.1). From the bottom figure, we can infer that, owing to the powerset-based generation of completeness statements, we observe an exponential growth of

relevant completeness statements. Still, the number of relevant statements drops drastically from the number of all completeness statements, thanks to the constant-relevance principle.

The incorporation of the RDFS increases the reasoning time, with the case of DBP queries being the worst. Nevertheless, on an absolute scale, it is still reasonably fast (i.e., on average, below 0.3 s), even when using relatively large ontologies with up to 7,000 RDFS axioms. To compare the results with plain reasoning, we also took for each case randomly 1,000 queries and measured the reasoning time with all completeness statements under RDFS semantics. The average runtimes were 236 ms for the SWDF case, 3,727 ms for the DBP case, and 498 ms for the LGD case. As such, our optimized technique can provide a speedup of up to 63.

Conclusions of the experiments. We have evaluated completeness reasoning in practical settings based on real query logs from DBpedia, SWDF, and LGD SPARQL endpoints. We observed that completeness reasoning with the constant-relevant principle can be done quickly, with the average time of submillisecond and the longest time of 159 ms. Compared with query evaluation time, completeness reasoning adds only a little overhead, just about 0.5%, on average. Also, the performance of completeness reasoning tends to be positively correlated with query length. Reasoning performance also relies on the number of constants, which, in practice, is relatively low (up to 14 constants). A possible weakness of our approach is when there are a large number of constants in the query (e.g., 32 constants) owing to the exponential blowup of the set-equality queries generated.

We also performed experiments of completeness reasoning under RDFS semantics. In this case, the completeness reasoning time increases, with the worst-case average of 285 ms for DBP queries. Completeness reasoning time with RDFS also positively correlates with the size of the used RDFS.

Regarding completeness reasoning for other query classes, the current results provide a reasonable hint. For the DISTINCT class, the only difference is that there is one additional projection operation, which has a negligible effect on the runtime (as per Theorem 3.6). Moreover, for the OPT class, from the statistics over the DBpedia query log, the average number of OPT operators in an OPT query is only 1.14, while 99% are below 3. Hence, according to Theorem 3.9, the reasoning time is roughly 4 times that without OPT, which would still be fast. Completeness reasoning in the federated and timestamped settings can also readily adopt the constant-relevance principle, with the additional little cost of data source identifier and timestamp bookkeeping. Consequently, the results of the experiments suggest that completeness reasoning can also be done reasonably well in those settings.

7 RELATED WORK

Data completeness, as defined by Wang and Strong (1996), is the *breadth*, *depth*, and *scope* of information contained in the data. Batini and Scannapieco (2016) considered data completeness to be one of the most significant data quality dimensions. Like other quality dimensions (e.g., accuracy and timeliness), the problem of data completeness may occur in various application domains, such as biology, aviation, and health care, as studied by Becker et al. (2015).

In the field of relational databases, concerns about data (in)completeness can be traced back to Codd (1979), who proposed a treatment of nulls based on 3-valued logic. Motro (1989) developed an integrity model for databases that considers completeness (and validity). Levy (1996) introduced local completeness statements, by which one can assert the completeness of parts of a database relation, and studied their relationship to relational query completeness. Razniewski and Nutt (2011) reduced the problem of query completeness to query containment and used this reduction to study the complexity of the completeness problem in the relational setting.

In the Semantic Web area, the problem of completeness is particularly challenging owing to the OWA. Several researchers studied completeness in the broader context of data quality. Fürber

and Hepp (2011) developed a generic vocabulary for data quality management in the Semantic Web. Their vocabulary can facilitate the standardized formulation of data quality rules, data quality problems, and data quality scores for RDF data sources. For example, one completeness-related problem that can be described is “missing element”: schema elements, instances, or property values are missing, when required. Mendes et al. (2012) proposed Sieve, a framework for Linked Data quality assessment and fusion. Sieve enables users to define quality scoring functions and perform conflict-resolution tasks based on the quality scores to combine RDF data from multiple sources. As an illustration, users can define a completeness scoring function based on the average number of properties of instances in a data source. A recent initiative to improve RDF data quality is underway by the W3C’s RDF Data Shapes group.²⁷ The group is developing SHACL (Knublauch and Kontokostas 2017), a language for validating RDF graphs against a set of conditions (called “shapes”). In SHACL, one can formulate integrity constraints, for example, by requiring that every person has a gender. The lack of such required information indicates incompleteness of data. By this approach, however, one cannot detect whether optional information, such as a spouse, is missing.

Zaveri et al. (2016) surveyed techniques to measure the completeness (among other data quality aspects) of RDF data sources. These techniques often measure completeness of a data source as the fraction of real-world information present in another data source that is chosen as the gold standard. The surveyed techniques were not concerned with how to express that a source is of gold-standard quality for some type of information. With our approach, one can describe the data quality aspect of completeness over RDF data sources, meaning that the sources can serve as the gold standard for parts captured by the statements. Harth and Speiser (2012) discussed the problem of assessing the completeness of Linked Data querying. They regarded the whole Web as the most ideal gold standard for evaluating queries. To be more realistic, they weakened that to data that is reachable from authoritative data sources. In their work, no assumption was made as to whether the whole Web really captures all information in the real world. Galárraga et al. (2013) stressed the need for complete information for rule mining over RDF KBs. Since completeness cannot be guaranteed, they introduced a partial completeness assumption (PCA) as a substitute, which states that if the KB knows some r -attribute of x , then it knows all r -attributes of x . Such an assumption is restricted in the sense that completeness is defined at the level of atomic attributes. Recently, Acosta et al. (2017) proposed HARE, a hybrid SPARQL engine to enhance the completeness degree of query answers. HARE implemented query execution strategies that can identify portions of queries yielding missing values and can perform microtask crowdsourcing to fill those missing values. As opposed to our work, HARE cannot be used to check whether queries are complete in the sense that all (ideal) answers are returned, as they focus rather on expanding the result of SPARQL queries.

In RDF, an existing way to close the scope of some data is via RDF collections, that is, groups of things represented as lists in the RDF graph (Manola and Miller 2004). Similarly, via enumerated classes, OWL provides a way to describe a closed class by enumerating all of its instances (Hitzler et al. 2012). Both approaches, however, consider only atomic classes as opposed to our more expressive approach, which leverages arbitrary BGPs to describe completeness (in which, for instance, the predicate position can be a variable). In Description Logics (DLs), which are a family of formalisms underlying OWL, several proposals have been made for partial closed-world features. Seylan et al. (2009) introduced the notion of a DBox whose behavior is similar to that of a database: that the predicate (i.e., concept and role) assertions in the DBox are closed (or complete). Lutz et al. (2013, 2015) generalized DBoxes by allowing a mix of open and closed predicates

²⁷<https://www.w3.org/2014/data-shapes>.

both in the schema and in the data (as opposed to DBoxes in which only closed predicates are permitted in data). Moreover, they provided a data complexity analysis of answering conjunctive queries (CQs) when closed predicates are admitted in DL KBs. Ngo et al. (2016) later complemented their work with a combined complexity analysis. All of the DL work above investigated a different problem than ours: given a DL KB with closed predicates, can certain facts be inferred and thus used for computing query answers? Our work studies the completeness property of queries, that is, whether queries can be answered completely over an RDF KB.

There have been proposals to add metadata to RDF resources. For providing provenance metadata, well-known vocabularies include Dublin Core Terms (DCMI 2012), MetaVocab,²⁸ and W3C PROV (Lebo et al. 2012). These vocabularies are intended to provide provenance information, such as creators, subjects, publishers, creation dates, and abstracts. While these vocabularies are generic, specialized vocabularies to describe (RDF and non-RDF) datasets exist. DCAT (Maali and Erickson 2014) is an RDF vocabulary for data catalogs, which can be of various formats, ranging from XML to RDF. Important DCAT concepts include Catalog, Dataset, and Distribution. VoID (Alexander et al. 2011) is more focused on describing RDF datasets. VoID covers four categories of metadata: general, access, structural, and description of links between datasets. With VoID it is possible, among other things, to provide information about how many instances a particular class has, the SPARQL endpoint of a source, and links to other data sources. While those approaches are more focused on provenance and quantitative information of datasets, our completeness metadata serves a different goal: to characterize datasets in terms of their completeness that is both conceptually well founded and practically applicable. Schmachtenberg et al. (2014) analyzed the adoption of metadata best practices over the Web of Linked Data. Some of the main findings are that around 37% of all datasets use provenance metadata and that VoID is used in around 15% of all datasets. The findings confirmed that metadata about RDF datasets is substantially deployed in practice. Thus, it is conceivable that statements about completeness can be added as metadata for RDF datasets as well—in particular, to complement the lack of formal completeness specifications in the existing proposals.

In Darari et al. (2013), we proposed a framework for managing completeness over RDF data sources and introduced the notions of completeness statements to describe complete parts of data sources and of query completeness. Moreover, we investigated the problem of completeness entailment: to check whether a set of completeness statements entails query completeness. Nevertheless, we expand this work by (i) providing a time extension, (ii) developing an indexing technique for completeness statements that can reduce the number of statements considered in the reasoning, and (iii) providing an experimental evaluation based on real SPARQL query logs to show the feasibility of query completeness reasoning.

Darari et al. (2016) formalized and characterized a case of completeness reasoning in which data-specific inferences are taken into account as opposed to the data-agnostic case (Darari et al. 2013). Furthermore, they proposed a practical class of completeness statements, called SP-statements, which consist of a single triple pattern and are used to state the completeness of a property of an entity. They developed an indexing technique for SP-statements, and conducted an experimental evaluation using the indexing.

8 DISCUSSION

In this section, we discuss relevant aspects of our completeness management framework, which are organized into two parts. The practical considerations part comprises modeling of completeness statements, correctness, availability and creation of completeness statements, data volatility,

²⁸<http://webns.net/mvcb/>.

and knowledge about ideal graphs. The technical considerations part comprises complexity, query coverage, FILTER extension, UNION extension, aggregate extension, blank nodes, and OWA and CWA interpretation of query answers.

Practical Considerations

Modeling of Completeness Statements. By their nature, completeness statements support capturing information of any kind as long as it can be represented by means of BGPs. The flexibility of BGPs allows us to express, for example, completeness over *multiple columns* (when the information comes from relational databases), by producing for each column predicate a corresponding completeness statement with that predicate. Completeness statements are also able to represent *multihop* information. As an illustration, the completeness statement for Trump’s daughters can be modeled as $Compl((trump, child, ?c), (?c, gender, female) \mid \emptyset)$. Here, we do not state that we are complete for all of Trump’s children, but instead all of Trump’s children whose gender is female (specified by a join). Similarly, completeness statements can naturally deal with *multi-valued properties*, e.g., the statement of completeness for Trump’s children refers to *all* his children.

Practically, publishers who want to provide data and consumers who want to query data would have to have some common understanding of which schema to use. We envision that when talking about the completeness of a data source, the statements that are created would use the schema or vocabulary of that data source.²⁹ This would reduce the learning curve in adopting completeness statements for both the data publisher and data consumer. Nevertheless, to foster the adoption, a simpler form of completeness statements might be useful. Darari et al. (2016) proposed SP-statements, a class of completeness statements suitable for crowdsourced, entity-centric KBs. SP-statements are of the form $Compl((s, p, ?v) \mid \emptyset)$, used to state the completeness of all values of the property p with respect to the KB entity s . This simplicity might potentially serve as a first step toward introducing completeness statements to the Linked Data community. SP-statements are used in COOL-WD (Darari et al. 2017) to provide completeness annotations over Wikidata, where users may add completeness statements by simply clicking the corresponding property box right inside Wikidata.

In the case in which data modeling violates schemas, we expect that such data might be difficult to use independently of the existence of completeness statements. Hence, in such situations, one might resort to standard approaches to detect such violations, for example, using SHACL (Knublauch and Kontokostas 2017) or ShEx (Prud’hommeaux et al. 2017) and then fixing the erroneous data using data cleaning tools such as the LOD Laundromat (Beek et al. 2014).

Correctness. Any inferences are only as correct as the used antecedents. If owners of data sources can add completeness annotations by themselves, incorrect completeness annotations may occur, which, in turn, may lead to incorrect conclusions. This issue cannot be avoided but can be made more transparent by annotating conclusions with information about the antecedents used (e.g., “conclusion is based on the completeness assertions X , Y , and Z over data source W , given by agents A and B on date D ”). Such provenance information can therefore serve as a basis for trust determination over conclusions. We refer to Artz and Gil (2007), Hartig (2009), and Lebo et al. (2012) for work about trust and provenance.

Another view on correctness is that as analogous to completeness statements, one can also formulate correctness statements and use them for annotating queries with correctness information. This was already observed by Motro (1989). While both completeness and correctness are

²⁹This applies also for data querying in which the query schema is generally the same as that of the data source’s schema.

important issues on the Semantic Web, we focus here on completeness, because we believe that correctness statements are less common.

Availability and Creation of Completeness Statements. At the core of the proposed framework lies the availability of completeness statements. We have discussed in Section 1.1 how existing data sources, such as IMDb, already incorporate such statements (Figure 1) and how they can be made machine readable using our framework (as in Section 2.2). Such availability of completeness statements rests on the assumption that a domain expert has the necessary background knowledge to provide completeness statements.

We believe that it is in the interest of data providers to annotate their data sources with statements about completeness in order to increase their value. Users can be more inclined to prefer data sources that include “completeness marks” to other data sources. For example, there were two tickets created for feature requests of data completeness in Wikidata.³⁰ Our completeness statement framework (particularly the RDF representation of completeness statements, as in Section 2.2) makes it explicit to both data publishers and data consumers what is meant by a data source that is “complete for some topic.” With completeness statements, both data publishers and consumers have a precise, shared understanding of what portions of data are complete. Not all topics may be given a crisp definition of completeness. Therefore, for some topics that can potentially be ambiguous, an additional textual description to provide context on completeness might be attached, for instance, as in the definition of completeness of cast and crew by IMDb.³¹ In the context of querying by data consumers, thanks to our completeness reasoning feature, layman-type consumers can still benefit from completeness statements since query results can be annotated in an automated way with completeness flags. Moreover, in the era of crowdsourcing, the availability of independent “ratings” from users can also contribute (as in Wikipedia and OpenStreetMap), in a bottom-up manner, to the description of the completeness of data sources. Systems to support the process of collaborative creation of completeness statements have been demonstrated by CORNER (Darari et al. 2014) and COOL-WD (Prasojo et al. 2016). CORNER³² focused on the development of a completeness statement hub for multiple RDF data sources. COOL-WD³³ features adding and viewing completeness statements directly from Wikidata, currently storing 10,000 completeness statements about Wikidata entities and showing a promising first step toward providing completeness metadata for Wikidata (and the Web of Linked Data in general).

A Web extraction technique based on manually created regular expressions was developed to extract cardinality information about children (Mirza et al. 2016). The technique was later generalized in Mirza et al. (2017), in which the authors proposed a distant-supervision method using conditional random fields to learn cardinality text patterns for four Wikidata relations (including child). Cardinalities extracted from their techniques can be leveraged to automatically generate completeness statements as follows: Whenever the cardinality matches the count of the corresponding relation values of an entity in an RDF data source, then a completeness statement for the entity’s relation can be generated. For instance, consider the text “Barack Obama has two children.” The cardinality information extracted would be of the form: $|(BarackObama, child)| = 2$. Now, when a data source contains two children of Obama, one can match this with the cardinality information and assert that the data source is complete for all children of Obama. Such a way of generating completeness statements might be a scalable alternative to manually generating completeness statements.

³⁰See, <https://phabricator.wikimedia.org/T150116> and <https://phabricator.wikimedia.org/T150116>.

³¹<https://help.imdb.com/article/contribution/filmography-credits/complete-cast-crew/GMZECKUS8JG34C4J>.

³²<http://corner.inf.unibz.it/>.

³³<http://cool-wd.inf.unibz.it/>.

Another way to add completeness statements is by using bots to import information from external authoritative sources (in the form of tables, or APIs) to RDF data sources. Bots are created manually, but the data import process can then be done automatically. To illustrate, the proceedings data of a conference is very likely to be complete for all the papers in the proceedings as well as their authors. Hence, when importing data, completeness statements about papers in proceedings and papers' authors can be added automatically using the corresponding bot. We refer the reader to an ongoing initiative to provide RDF data about Semantic Web proceedings, the ScholarlyData project,³⁴ in which completeness statements can be potentially added automatically.

As for providing timestamps for completeness statements, several options exist, such as adding explicitly timestamps to completeness statements, as in Wikipedia,³⁵ or using the "last updated" information for the respective data.

Data Volatility. Batini and Scannapieco (2016) distinguish two types of data elements based on the frequency of changes: (i) stable data elements and (ii) volatile data elements. As for stable data elements, nontimestamped completeness statements (as in Section 2.1) are adequate to capture their completeness. As for volatile data elements, a different strategy is needed to make sure that completeness statements do not become outdated when the data changes. We have incorporated this strategy into timestamped completeness statements (as in Section 5). Those statements provide a boundary up to when the corresponding data is complete, which relaxes the need for immediate updates of completeness statements. Additionally, when the corresponding data captured by the timestamped statements is also annotated with timestamps, our framework may provide exact (i.e., sound and complete) answers for queries with respect to some specified point in time.

Knowledge about Ideal Graphs. While the ideal graph as a whole is difficult (if not impossible) to obtain in practice, knowledge about portions of the ideal graph is available in many different forms.

One prime form is authoritative sources. Government, company, and university websites (among others) provide primary information about themselves, which is often complete. For instance, the official Germany portal [deutschland.de](https://www.deutschland.de) lists all 16 federal states in Germany,³⁶ Schneider Electric lists all of its products,³⁷ and the website of TU Dresden lists all of its schools and faculties.³⁸ In addition to authoritative sources, knowledge about the ideal graph may come from websites that already contain (natural language) completeness annotations, such as Wikipedia, IMDb, and OpenStreetMap (as in Section 1.1). In this case, the ideal graph knowledge relies on the wisdom of the crowd. Other forms of knowledge about the ideal graph may include research papers (e.g., the complete list of Hilbert's Mathematical Problems (Hilbert 1902), books (e.g., the Harry Potter book series for (complete) knowledge about Harry Potter), patents, court cases, and speech transcripts. Basically, any sources that may give information may also provide completeness information (i.e., knowledge about the ideal graph).

Technical Considerations

Complexity. All problems of checking completeness entailment were shown to be NP-complete in combined complexity. This is good news, as it says that completeness checking is no more difficult than query evaluation. In particular, even though considering bag semantics for queries,

³⁴<http://www.scholarlydata.org/>.

³⁵https://en.wikipedia.org/wiki/Template:Complete_list.

³⁶<https://www.deutschland.de/en/topic/politics/germany-europe/state-governments>.

³⁷<https://www.schneider-electric.co.id/en/all-products/>.

³⁸https://tu-dresden.de/tu-dresden/organisation/bereiche-fakultaeten?set_language=en.

where query containment is Π_p^2 -hard (Chaudhuri and Vardi 1993), we retain the complexity of set semantics.

Query Coverage. SPARQL queries may also contain other operators, such as FILTER, UNION, and so on. Nevertheless, the use of SPARQL queries in practice is dominated by the operators AND and OPT, whose completeness checking procedure was proposed in Section 3. Our analysis from the Linked SPARQL Queries (LSQ) dataset, which contains query logs from, among others, DBpedia, has shown that around 3 out of 4 queries asked over DBpedia were about AND and/or OPT, without other operators.

FILTER Extension. Our proposed framework provides a sufficient characterization for completeness entailment of queries with FILTER. The idea is that, for any query with FILTER, it can be answered completely if completeness statements can guarantee the completeness of the query where the FILTER parts are removed. Having a full characterization with the incorporation of FILTER in both completeness statements and queries is left for future work.

UNION Extension. For queries with nonnested UNION, completeness can be guaranteed iff all the BGPs of the UNION parts can be guaranteed to be complete. For queries with nested UNION, a flattening transformation to DNF transformation of propositional formulas (Ligeza 2006) can be employed first, which generates an equivalent query with nonnested UNION. This approach, however, may potentially lead to an exponential blowup of the query size.

Aggregate Extension. SPARQL 1.1 includes aggregate operators such as COUNT, SUM, and MAX. Computing aggregates over incomplete data may produce incorrect results. Our completeness checking technique can be leveraged to check whether the body of the aggregate queries can be guaranteed to be complete. If the answer is yes, then the correctness of the resulting aggregate values can be ensured.

Blank Nodes. Blank nodes provide support for anonymous resources in RDF (Klyne and Carroll 2004). Owing to the local scope of their labels, blank nodes may add complexity to data processing (Erxleben et al. 2014; Heath and Bizer 2011). Moreover, semantic mismatches may occur when blank nodes are used in conjunction with, for example, SPARQL's COUNT (Hogan et al. 2014). Despite this, they are used in practice, to some extent, for modeling unknown nulls (Darari et al. 2015; Hogan et al. 2014) and n -ary relations (Noy and Rector 2006). The use of blank nodes for unknown nulls contradicts the idea of completeness: one may state that a graph is complete for triples of the form $(tom, spouse, ?s)$, while the graph contains the triple $(tom, spouse, _:b)$, indicating that Tom has a spouse who is unknown. For the case of n -ary relations, skolemization can be leveraged as a way to systematically replace blank nodes with fresh, skolem IRIs (Cyganiak et al. 2014; Hayes and Patel-Schneider 2014; Hogan 2015). Such a practice is followed, for example, by Wikidata developers to avoid blank nodes' complications (Erxleben et al. 2014).³⁹ This way, completeness statements can still represent the information previously encoded with blank nodes.

OWA and CWA Interpretation of Query Answers. RDF data is usually interpreted under the open-world assumption (OWA), for which it is unknown whether the information is complete or not (Patel-Schneider 2015). The presence of completeness statements adds knowledge about the fact that *some* portions of RDF datasets are known to be closed. Consequently, SPARQL queries evaluated *only* over those (closed) portions are guaranteed to return complete answers. As an illustration, consider the query "Retrieve movies directed by Tarantino." Without completeness

³⁹For instance, the IRI of Wikidata for Barack Obama's presidency is <http://www.wikidata.org/entity/statement/q30-14c52ecf-4b9e-083b-86c8-86029c334d99>.

statements, answers to this query are always open for possible extensions since the data source upon which the query is evaluated might miss some Tarantino movies. Now, when we know that the data source is complete for all Tarantino movies, a corresponding completeness statement may be formulated that makes this knowledge explicit. As an effect, answers to the query for Tarantino movies are now closed for extensions (as guaranteed by the statement); hence, those answers are complete. Yet, answers to other queries whose data is not captured by completeness statements are still open. Thus, we can say that our approach stands in the middle between the classic open- and closed-world assumptions.

9 CONCLUSIONS AND FUTURE WORK

In this article, we have presented a theoretical framework to manage completeness of RDF data sources. The framework focuses on providing formal and machine-readable completeness descriptions (or statements) about RDF data sources and reasoning methods to infer whether query completeness can be guaranteed given such descriptions. Such an automated way to infer query completeness was not possible before, since (i) completeness descriptions were provided only in natural language, such as those found on IMDb, Wikipedia, and OpenStreetMap sites; and (ii) it was not clear how completeness descriptions can guarantee query completeness.

Our completeness management framework supports a variety of scenarios to capture different requirements for managing completeness. The first and simplest scenario is when only a single data source is considered to entail query completeness. Here, we cover several query classes: basic queries, DISTINCT queries, queries with the OPT (“optional”) keyword, and queries under RDFS semantics. A more complex scenario is when completeness statements of multiple data sources are taken into account in completeness reasoning. We presented a federation extension to enable rewriting of a complete query into a federated one, which assigns each query part to a suitable complete source. Another scenario is when data dynamicity over time is also an issue. We presented a time extension to represent more expressive completeness statements with timestamps to bound their completeness scope. Hence, in addition to checking whether query answers are complete, we can also check the guaranteed completeness date of the query, that is, the latest date for which the query completeness can be guaranteed.

Real-world RDF data sources, such as DBpedia, may contain a large amount of data. Consequently, to describe their completeness one would need a large number of completeness statements. We presented a technique for efficient completeness reasoning over large sets of statements based on the constant-relevance principle to rule out a significant number of irrelevant statements in completeness reasoning. We developed a retrieval technique for constant-relevant statements based on the standard hashing. From our experimental evaluations, we concluded that completeness reasoning added only a little overhead to query evaluation.

For future work, we are interested in investigating the relationship between our completeness reasoning framework and the OWL ontology languages. Our conjecture is that the OWL 2 RL profile (Hitzler et al. 2012) is suitable to be integrated with our framework owing to its rule-based axiomatization. It is also in our interest to gain deeper insights into how in practice (natural language) completeness statements are produced and consumed by Web communities. Inspired by such insights, we may then devise practical guidelines and workflows for a more effective and robust completeness management for RDF data sources. Moreover, to have an impact on the wider community, another future direction is to conduct extensive case studies about the completeness aspect of data quality in various application domains, such as life sciences, government, and media. The main goal here would be to analyze whether our completeness framework is sufficient to support their data completeness requirements and, if not, to determine which extensions are needed. Extending our framework to support queries with negation is also in our plan. This is especially

important since evaluating queries with negation is usually done in a closed-world manner in contrast to RDF itself, which is often interpreted under the OWA. Therefore, the question is how one can perform negation safely (in terms of query soundness) by also incorporating data completeness.

APPENDIXES

A PROOF OF THEOREM 3.3

THEOREM 3.3 (COMPLETENESS OF BASIC QUERIES). *Let C be a set of completeness statements and let $Q = (W, P)$ be a basic query. Then,*

$$C \models \text{Compl}(Q) \quad \text{iff} \quad \tilde{P} = T_C(\tilde{P}).$$

PROOF. “ \Rightarrow ” We prove the contrapositive. We show that, if $\tilde{P} \neq T_C(\tilde{P})$, then the incomplete data source $(T_C(\tilde{P}), \tilde{P})$ serves as a counterexample for the entailment $C \models \text{Compl}(Q)$. It satisfies C by Proposition 2.5 but does not satisfy $\text{Compl}(Q)$. The reason is that the freeze mapping \tilde{id} cannot be retrieved when evaluating P over the available graph $T_C(\tilde{P})$, while it can over the ideal graph \tilde{P} .

“ \Leftarrow ” Assume that $\tilde{P} = T_C(\tilde{P})$. We will now prove that, for every incomplete data source $\mathcal{G} = (G^a, G^i)$ satisfying C , it holds that $\mathcal{G} \models \text{Compl}(Q)$. It is sufficient to prove that $\llbracket P \rrbracket_{G^a} = \llbracket P \rrbracket_{G^i}$. Observe that $\llbracket P \rrbracket_{G^a} \subseteq \llbracket P \rrbracket_{G^i}$ follows immediately from the monotonicity of P and the fact that $G^a \subseteq G^i$ by definition.

Now, we prove that $\llbracket P \rrbracket_{G^a} \supseteq \llbracket P \rrbracket_{G^i}$. Consider a mapping $\mu \in \llbracket P \rrbracket_{G^i}$. We want to show that $\mu \in \llbracket P \rrbracket_{G^a}$. From $\mu \in \llbracket P \rrbracket_{G^i}$, it follows that $\mu P \subseteq G^i$. Due to the monotonicity of T_C , it is the case that $T_C(\mu P) \subseteq T_C(G^i)$. The inclusion can be extended to $T_C(\mu P) \subseteq T_C(G^i) \subseteq G^a$ due to $(G^a, G^i) \models C$. From the assumption that $\tilde{P} = T_C(\tilde{P})$ and the prototypicality of \tilde{P} , it follows that $\mu \tilde{id}^{-1} \tilde{P} \subseteq T_C(\mu \tilde{id}^{-1} \tilde{P})$, where \tilde{id}^{-1} is the inverse of the freeze mapping \tilde{id} . Note that $\mu \tilde{id}^{-1} \tilde{P} = \mu P$. Thus, it holds that $\mu P \subseteq T_C(\mu P)$, and from $T_C(\mu P) \subseteq G^a$, it follows that $\mu P \subseteq G^a$. Consequently, μ is also in $\llbracket P \rrbracket_{G^a}$. Since μ was arbitrary, we proved that $\llbracket P \rrbracket_{G^a} \supseteq \llbracket P \rrbracket_{G^i}$. \square

B PROOF OF COROLLARY 3.4

COROLLARY 3.4. *Deciding whether $C \models \text{Compl}(Q)$, given a set C of completeness statements and a basic query $Q = (W, P)$, is NP-complete.*

PROOF. The NP-membership holds because we can verify in polynomial time that $\tilde{P} = T_C(\tilde{P})$ (see Theorem 3.3) by guessing for each triple in \tilde{P} , the query Q_C , and the mapping by which it is constructed.

The proof of NP-hardness is by a reduction of the 3-colorability problem of directed graphs, which is NP-hard. We adopt to our setting the reduction by Chandra and Merlin (1977) of graph-coloring to query evaluation.

We first encode a directed graph $G = (V, E)$, for which we want to check whether it is 3-colorable, as a set $\text{triples}(G)$ of triple patterns. We associate to each vertex $v \in V$ a new variable $?v$, and then create a new IRI *edge*. Then, we define $\text{triples}(G)$ as the set of all triple patterns $(?v_i, \text{edge}, ?v_j)$ created from each pair $(v_i, v_j) \in E$. Next, we create the completeness statement C_G :

$$\text{Compl}(\text{triples}(G) \cup \{ (r, \text{edge}, g), (r, \text{edge}, b), (g, \text{edge}, r), (g, \text{edge}, b), (b, \text{edge}, r), (b, \text{edge}, g) \} \mid \emptyset).$$

Now, consider the query Q_{col} :

$$\{ \{ \}, \{ (r, \text{edge}, g), (r, \text{edge}, b), (g, \text{edge}, r), (g, \text{edge}, b), (b, \text{edge}, r), (b, \text{edge}, g) \} \}.$$

Then, the following claim holds:

$$\text{The directed graph } G \text{ is 3-colorable} \quad \text{iff} \quad \{C_G\} \models \text{Compl}(Q_{col}).$$

PROOF OF THE CLAIM: “ \Rightarrow ” Suppose that G is 3-colorable. Then, there is a mapping μ from the vertices of G to set of three colors $\{r, g, b\}$ such that no adjacent nodes have the same color. It therefore follows that $\{C_G\} \models \text{Compl}(Q_{col})$ by construction.

“ \Leftarrow ” We will prove the contrapositive. Assume that G is not 3-colorable. Then, there is no mapping from the vertices of G to the set $\{r, g, b\}$ such that each pair of adjacent nodes has different colors. Consider the incomplete data source $\mathcal{G} = (G^a, G^i)$, where $G^a = \emptyset$ and $G^i = \{(r, \text{edge}, g), (r, \text{edge}, b), \dots, (b, \text{edge}, g)\}$. By the construction of C_G and Q_{col} , we have that $\mathcal{G} \models \{C_G\}$, but $\llbracket Q_{col} \rrbracket_{G^a} \neq \llbracket Q_{col} \rrbracket_{G^i}$. This implies that there is a counterexample for $\{C_G\} \models \text{Compl}(Q_{col})$. \square

C PROOF OF THEOREM 3.6

THEOREM 3.6 (COMPLETENESS OF DISTINCT QUERIES). *Let C be a set of completeness statements and let $Q = (W, P)^d$ be a DISTINCT query. Then,*

$$C \models \text{Compl}(Q) \quad \text{iff} \quad \pi_W(\tilde{id}) \in \llbracket Q \rrbracket_{T_C(\tilde{P})}.$$

PROOF. “ \Rightarrow ” We prove the contrapositive. We show that if $\pi_W(\tilde{id}) \notin \llbracket Q \rrbracket_{T_C(\tilde{P})}$, then the incomplete data source $(T_C(\tilde{P}), \tilde{P})$ is a counterexample for the entailment $C \models \text{Compl}(Q)$. It satisfies C by Proposition 2.5 but does not satisfy $\text{Compl}(Q)$. The reason is that the mapping $\pi_W(\tilde{id})$ cannot be retrieved by Q over the available graph $T_C(\tilde{P})$, while it can over the ideal graph \tilde{P} .

“ \Leftarrow ” Assume that $\pi_W(\tilde{id}) \in \llbracket Q \rrbracket_{T_C(\tilde{P})}$. We will prove that, for every incomplete data source $\mathcal{G} = (G^a, G^i)$ satisfying C , it holds that $\mathcal{G} \models \text{Compl}(Q)$. Observe that $\llbracket Q \rrbracket_{G^a} \subseteq \llbracket Q \rrbracket_{G^i}$ follows immediately from the monotonicity of Q and $G^a \subseteq G^i$.

Now, we prove that $\llbracket Q \rrbracket_{G^a} \supseteq \llbracket Q \rrbracket_{G^i}$. Suppose that there is a mapping $\mu \in \llbracket Q \rrbracket_{G^i}$. We want to show that $\mu \in \llbracket Q \rrbracket_{G^a}$. From our assumption that $\pi_W(\tilde{id}) \in \llbracket Q \rrbracket_{T_C(\tilde{P})}$, it follows that there is a mapping μ_{ext} where $\mu \subseteq \mu_{ext}$ and $\mu_{ext}P \subseteq G^i$ such that $\mu \tilde{id}^{-1} \pi_W(\tilde{id}) \in \llbracket Q \rrbracket_{T_C(\mu_{ext} \tilde{id}^{-1} \tilde{P})}$. Observe that $\mu \tilde{id}^{-1} \pi_W(\tilde{id}) = \mu$ and $\mu_{ext} \tilde{id}^{-1} \tilde{P} = \mu_{ext}P$. This means that we are able to preserve by T_C some graph (e.g., $T_C(\mu_{ext}P)$) that contributes to the mapping μ .

Because of the monotonicity of T_C , it holds that $T_C(\mu_{ext}P) \subseteq T_C(G^i)$. The inclusion can further be extended to $T_C(\mu_{ext}P) \subseteq T_C(G^i) \subseteq G^a$ due to the assumption that $(G^a, G^i) \models C$. This means that $T_C(\mu_{ext}P) \subseteq G^a$ and, therefore, $\mu \in \llbracket Q \rrbracket_{G^a}$. Since μ was arbitrary, we proved that $\llbracket Q \rrbracket_{G^a} \supseteq \llbracket Q \rrbracket_{G^i}$. \square

D PROOF OF THEOREM 3.9

For a pattern tree $\mathcal{T} = ((N, E, r), \mathcal{P})$, a node n in \mathcal{T} , and the parent node \hat{n} of n , we define the operator

$$\text{newvar}(n) = \text{var}(\mathcal{P}(n)) \setminus \text{var}(\mathcal{P}(\hat{n})),$$

which returns all variables occurring in the BGP of n , but not in its parent. We use the following proposition from Letelier et al. (2012) to prove the theorem.

PROPOSITION (NEW VARIABLES). *Let \mathcal{T} be a QWDPT in NR normal form. Then, for every node n in \mathcal{T} that is not the root, it holds that $\text{newvar}(n) \neq \emptyset$.*

Furthermore, Letelier et al. (2012) defined the top-down evaluation $\llbracket \mathcal{T} \rrbracket_G^{td}$ of a QWDPT \mathcal{T} of a well-designed OPT query Q and proved it to be equivalent to $\llbracket Q \rrbracket_G$.

Definition (Top-Down Evaluation of QWDPTs). Let G be a graph, $\mathcal{T} = ((N, E, r), \mathcal{P})$ be a QWDPT, and M be a set of mappings. For $n \in N$, the evaluation of \mathcal{T}_n (the complete subtree of \mathcal{T} rooted at n) with regard to M and G , written $ext(M, n, G)$, is defined as follows: If n is a leaf, then

$$ext(M, n, G) = M \bowtie \llbracket \mathcal{P}(n) \rrbracket_G,$$

and, otherwise, if n_1, \dots, n_k are the children of n , then

$$ext(M, n, G) = M_1 \bowtie M_2 \bowtie \dots \bowtie M_k,$$

where $M_i = (M \bowtie \llbracket \mathcal{P}(n) \rrbracket_G) \bowtie ext(M \bowtie \llbracket \mathcal{P}(n) \rrbracket_G, n_i, G)$. The *top-down evaluation* of \mathcal{T} over G , written $\llbracket \mathcal{T} \rrbracket_G^{td}$, is defined as

$$\llbracket \mathcal{T} \rrbracket_G^{td} = ext(\{\emptyset\}, r, G).$$

THEOREM 3.9 (COMPLETENESS OF OPT QUERIES). *Let C be a set of completeness statements, let Q be a well-designed OPT query, and let \mathcal{T} be an equivalent QWDPT of Q in NR normal form. Then,*

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

PROOF. “ \Rightarrow ” We prove the contrapositive. We show that if there is a branch query $Q_k = (var(P_k), P_k)$ of \mathcal{T} where $C \not\models Compl(Q_k)$, then the incomplete data source $(T_C(\tilde{P}_k), \tilde{P}_k)$ is a counterexample for $C \models Compl(Q)$. It satisfies C by Proposition 2.5, but does not satisfy $Compl(Q)$. The reason is that the freeze mapping \tilde{id}_k of P_k can be retrieved by evaluating Q over the ideal graph \tilde{P}_k but not over the available graph $T_C(\tilde{P}_k)$. Note that the proposition “New Variables” and \mathcal{T} being a QWDPT in NR normal form prevent Q from binding the mapping \tilde{id}_k by other OPT patterns in Q than the corresponding OPT pattern of the branch query Q_k .

“ \Leftarrow ” We proceed by induction on the number of nodes, which is the same as the number of branch queries:

Induction Base: Suppose that there is only one node. Thus, the implication follows immediately from Theorem 3.3, as there is only one branch query of \mathcal{T} .

Induction Hypothesis: The implication holds for QWDPTs with k nodes.

Induction Step: We now prove that if the number of nodes is $k + 1$, the implication holds: If $C \models Compl(Q_i)$ for all branch queries Q_i of \mathcal{T} where $1 \leq i \leq k + 1$, then $C \models Compl(Q)$.

Let \mathcal{T}' be a QWDPT from the removal of a leaf node n_l from \mathcal{T} . From our assumption, we have that $C \models Compl(Q_i)$ for all branch queries Q_i of \mathcal{T}' . Note that the number of nodes in \mathcal{T}' is k . By the induction hypothesis, this implies that $C \models Compl(Q')$, where Q' is the corresponding query of \mathcal{T}' . With respect to top-down evaluation, this also means that $\llbracket \mathcal{T}' \rrbracket_{G^a}^{td} = \llbracket \mathcal{T}' \rrbracket_{G^i}^{td}$ for all $G = (G^a, G^i) \models C$.

Note that \mathcal{T} can be recreated by putting the node n_l as a child of its respective parent \hat{n}_l in \mathcal{T}' . Suppose that the children of \hat{n}_l are $n_1, \dots, n_{l-1}, n_l, n_{l+1}, \dots, n_j$. Then, given a set M of mappings and a graph G , the evaluation of $\mathcal{T}_{\hat{n}_l}$ is as follows:

$$ext(M, \hat{n}_l, G) = M_1 \bowtie \dots \bowtie M_{l-1} \bowtie M_l \bowtie M_{l+1} \bowtie \dots \bowtie M_j.$$

From the definition of the ext operator and n_l being a leaf in \mathcal{T} , it follows that

$$M_l = (M \bowtie \llbracket \mathcal{P}(\hat{n}_l) \rrbracket_G) \bowtie (M \bowtie \llbracket \mathcal{P}(\hat{n}_l) \rrbracket_G \bowtie \llbracket \mathcal{P}(n_l) \rrbracket_G).$$

Note that, by the definition of top-down evaluation, the set $ext(M, \hat{n}_l, G)$ is the source of the difference between the top-down evaluation of \mathcal{T} and that of \mathcal{T}' . The reason is that in \mathcal{T} we have to also consider the set M_l of mappings from the node n_l . Thus, if we are complete for this, we are complete for the whole query of the QWDPT \mathcal{T} .

By the induction hypothesis on \mathcal{T}' , for all incomplete data sources (G^a, G^i) satisfying C , it holds that $ext(M, \hat{n}_l, G^a)$ evaluated during the computation of $\llbracket \mathcal{T}' \rrbracket_{G^a}^{td}$ is equivalent to $ext(M, \hat{n}_l, G^i)$

evaluated during the computation of $\llbracket \mathcal{T}' \rrbracket_{G^i}^{td}$. Note that with regard to \mathcal{T}' , the value of $ext(M, \hat{n}_l, G^a)$ is $M_1 \bowtie \dots \bowtie M_{l-1} \bowtie M_{l+1} \bowtie \dots \bowtie M_j$, that is, there is no M_l . From our assumption that $C \models Compl(Q_i)$ for all branch queries Q_i of \mathcal{T}' , it is also the case that $M \bowtie \llbracket \mathcal{P}(\hat{n}_l) \rrbracket_{G^a} = M \bowtie \llbracket \mathcal{P}(\hat{n}_l) \rrbracket_{G^i}$.

Recall our assumption that $C \models Compl(Q_i)$, where $1 \leq i \leq k+1$. This implies that $M \bowtie \llbracket \mathcal{P}(\hat{n}_l) \rrbracket_{G^a} \bowtie \llbracket \mathcal{P}(n_l) \rrbracket_{G^a} = M \bowtie \llbracket \mathcal{P}(\hat{n}_l) \rrbracket_{G^i} \bowtie \llbracket \mathcal{P}(n_l) \rrbracket_{G^i}$. Thus, it also holds that $ext(M \bowtie \llbracket \mathcal{P}(\hat{n}_l) \rrbracket_{G^a}, n_l, G^a) = ext(M \bowtie \llbracket \mathcal{P}(\hat{n}_l) \rrbracket_{G^i}, n_l, G^i)$ and, therefore, we are complete for M_l . Consequently, $ext(M, \hat{n}_l, G^a) = ext(M, \hat{n}_l, G^i)$ on the top-down evaluation of \mathcal{T} over G^a and G^i , respectively. Since $\mathcal{T} \equiv Q$, it is the case that we are also complete for Q . Thus, we conclude that $C \models Compl(Q)$. \square

E PROOF OF THEOREM 3.13

THEOREM 3.13 (COMPLETENESS UNDER RDFS). *Let C be a set of completeness statements, let $Q = (W, P)$ be a basic query, and let S be a schema graph. Then,*

$$C \models_S Compl(Q) \quad \text{iff} \quad \tilde{P} \subseteq T_C^S(\tilde{P}).$$

PROOF. “ \Rightarrow ” We prove the contrapositive. We show that if $\tilde{P} \not\subseteq T_C^S(\tilde{P})$, then the incomplete data source $(T_C^S(\tilde{P}), cl_S(\tilde{P}))$ is a counterexample for the entailment. By the definition of T_C^S , it satisfies C with regard to the schema S , but does not satisfy $Compl(Q)$. The reason is that the freeze mapping \tilde{id} cannot be retrieved by P over the available graph $T_C^S(\tilde{P})$, while it can over the ideal graph $cl_S(\tilde{P})$. “ \Leftarrow ” Assume that $\tilde{P} \subseteq T_C^S(\tilde{P})$. We will prove that, for every incomplete data source $\mathcal{G} = (cl_S(G^a), cl_S(G^i))$ satisfying C , it holds that $\mathcal{G} \models Compl(Q)$. By definition, $\mathcal{G} \models Compl(Q)$ if $\llbracket Q \rrbracket_{cl_S(G^a)} = \llbracket Q \rrbracket_{cl_S(G^i)}$. It is sufficient to prove that $\llbracket P \rrbracket_{cl_S(G^a)} = \llbracket P \rrbracket_{cl_S(G^i)}$. Observe that $\llbracket P \rrbracket_{cl_S(G^a)} \subseteq \llbracket P \rrbracket_{cl_S(G^i)}$ follows immediately from the monotonicity of P and the fact that $cl_S(G^a) \subseteq cl_S(G^i)$.

As for $\llbracket P \rrbracket_{cl_S(G^a)} \supseteq \llbracket P \rrbracket_{cl_S(G^i)}$, suppose that there is a mapping $\mu \in \llbracket P \rrbracket_{cl_S(G^i)}$. This implies that $\mu P \subseteq cl_S(G^i)$. Because of the monotonicity of T_C^S , it holds that $T_C^S(\mu P) \subseteq T_C^S(G^i)$. By definition, $(cl_S(G^a), cl_S(G^i)) \models C$ implies that $T_C^S(cl_S(G^i)) \subseteq cl_S(G^a)$. By applying the closure cl_S once again on both sides of the inclusion, we have that $T_C^S(G^i) \subseteq cl_S(G^a)$. Thus, we have the inclusions $T_C^S(\mu P) \subseteq T_C^S(G^i) \subseteq cl_S(G^a)$.

From our assumption that $\tilde{P} \subseteq T_C^S(\tilde{P})$, it follows that $\mu \tilde{id}^{-1} \tilde{P} \subseteq \mu \tilde{id}^{-1} T_C^S(\tilde{P}) \subseteq T_C^S(\mu \tilde{id}^{-1} \tilde{P})$. Since $\mu \tilde{id}^{-1} \tilde{P} = \mu P$ and $T_C^S(\mu \tilde{id}^{-1} \tilde{P}) = T_C^S(\mu P) \subseteq cl_S(G^a)$, it holds that $\mu P \subseteq cl_S(G^a)$. Consequently, μ is also in $\llbracket P \rrbracket_{cl_S(G^a)}$. Because of the arbitrariness of the mapping μ , it is the case that $\llbracket P \rrbracket_{cl_S(G^a)} \supseteq \llbracket P \rrbracket_{cl_S(G^i)}$. \square

F PROOF OF COROLLARY 3.14

COROLLARY 3.14. *Deciding whether $C \models_S Compl(Q)$, given a set C of completeness statements, a schema graph S , and a basic query Q , is NP-complete.*

PROOF. From Theorem 3.13, it is the case that $C \models_S Compl(Q)$ iff $\tilde{P} \subseteq T_C^S(\tilde{P})$. Recall that the operator $T_C^S(\tilde{P})$ can be unfolded into $cl_S(T_C^S(cl_S(\tilde{P})))$. From Muñoz et al. (2009), the closure computation can be done in PTIME, with the size of the closure growing polynomially. We now provide an NP-procedure to check $C \models_S Compl(Q)$. First, we compute $cl_S(\tilde{P})$. Then, for every triple (s, p, o) in \tilde{P} , we guess a mapping and a CONSTRUCT query Q_C over $cl_S(\tilde{P})$ to build a graph whose closure computation contains the triple (s, p, o) .

The NP-hardness follows immediately because completeness entailment with RDFS is a generalization of the one without RDFS. \square

G PROOF OF PROPOSITION 4.9

PROPOSITION 4.9. *Let \bar{C} be a set of indexed completeness statements and let Q be a basic query. Then,*

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

PROOF. “ \Rightarrow ” Assume that $\bar{C} \models_{fed} Compl(Q)$ holds. Consider an incomplete data source $\mathcal{G} = (G^a, G^i)$ such that $(G^a, G^i) \models \bar{C}^{fl}$. We will prove that $(G^a, G^i) \models Compl(Q)$. Let us build an incomplete FDS (\bar{G}^a, \bar{G}^i) in which all graphs in \bar{G}^a are G^a . It follows that $(\bar{G}^a, \bar{G}^i) \models_{fed} \bar{C}$ and, thus, $(\bar{G}^a, \bar{G}^i)^{fl} \models Compl(Q)$ from the assumption that $\bar{C} \models_{fed} Compl(Q)$. Since $(\bar{G}^a, \bar{G}^i)^{fl} = (G^a, G^i)$ by construction, it holds that $(G^a, G^i) \models Compl(Q)$. Therefore, $\bar{C}^{fl} \models Compl(Q)$ is true, since \mathcal{G} was arbitrary.

“ \Leftarrow ” Assume that $\bar{C}^{fl} \models Compl(Q)$ is true. Take any incomplete FDS $\bar{\mathcal{G}}$ such that $\bar{\mathcal{G}} \models_{fed} \bar{C}$. We will prove that $\bar{\mathcal{G}}^{fl} \models Compl(Q)$. By the definition of flattening, if $\bar{\mathcal{G}} \models_{fed} \bar{C}$ holds, then $\bar{\mathcal{G}}^{fl} \models \bar{C}^{fl}$ also holds. Because of the assumption, it is the case that $\bar{\mathcal{G}}^{fl} \models Compl(Q)$. Therefore, we proved that $\bar{C} \models_{fed} Compl(Q)$ since $\bar{\mathcal{G}}$ was arbitrary. \square

H PROOF OF PROPOSITION 4.13

PROPOSITION 4.13. *Let \bar{C} be a set of indexed completeness statements and let $Q = (W, P)$ be a basic query. Then,*

$$\bar{C} \models_{fed} Compl(Q) \quad \text{iff} \quad \tilde{P} = (T_{\bar{C}}(\tilde{P}))^{fl}.$$

PROOF. By Proposition 4.9, it is the case that $\bar{C} \models_{fed} Compl(Q)$ iff $\bar{C}^{fl} \models Compl(Q)$. By Theorem 3.3, it holds that $\bar{C}^{fl} \models Compl(Q)$ iff $\tilde{P} = T_{\bar{C}^{fl}}(\tilde{P})$. Therefore, it is sufficient to prove that $T_{\bar{C}^{fl}}(\tilde{P}) = (T_{\bar{C}}(\tilde{P}))^{fl}$. However, this follows immediately from the definition of flattening. \square

I PROOF OF LEMMA 4.17

LEMMA 4.17. *Let \bar{C} be a set of indexed completeness statements and let $Q = (W, P)$ be a basic query such that $\bar{C} \models_{fed} Compl(Q)$. Moreover, let \bar{S} be a smart set with regard to \bar{C} and Q , and let $P_{\bar{S}}$ be the SERVICE transformation from the smart set \bar{S} . Then,*

$$\bar{Q} = (W, P_{\bar{S}}) \text{ is a smart rewriting of } Q \text{ with regard to } \bar{C} \text{ such that } \bar{C} \models_{fed} Compl(\bar{Q}).$$

PROOF. To prove that \bar{Q} is a smart rewriting, we will show that $\llbracket \bar{Q} \rrbracket_{(j_0, \bar{G}^a)} = \llbracket Q \rrbracket_{\cup_{j \in J} G_j^a}$ for every (\bar{G}^a, G^i) satisfying \bar{C} and every $j_0 \in J$. Consider an incomplete FDS $\bar{\mathcal{G}} = (\bar{G}^a, G^i)$ satisfying \bar{C} and an IRI $j_0 \in J$. From the assumption that $\bar{C} \models_{fed} Compl(Q)$, it follows that $\llbracket Q \rrbracket_{\cup_{j \in J} G_j^a} = \llbracket Q \rrbracket_{G^i}$. This means that instead we can prove that $\llbracket Q \rrbracket_{G^i} = \llbracket \bar{Q} \rrbracket_{(j_0, \bar{G}^a)}$. Moreover, it is sufficient to prove $\llbracket P \rrbracket_{G^i} = \llbracket P_{\bar{S}} \rrbracket_{(j_0, \bar{G}^a)}$. Note that proving this implies that $\bar{C} \models_{fed} Compl(\bar{Q})$ since $(P_{\bar{S}})^{fl} = P$ is true.

Let us first prove that $\llbracket P \rrbracket_{G^i} \subseteq \llbracket P_{\bar{S}} \rrbracket_{(j_0, \bar{G}^a)}$. By construction, it is the case that $(P_{\bar{S}})^{fl} = P$. We will prove that, for every (SERVICE k t) occurring in $P_{\bar{S}}$ such that k is an IRI and t is a triple pattern, it holds that $\pi_{var(t)}(\llbracket P \rrbracket_{G^i}) \subseteq \llbracket t \rrbracket_{G_k^a}$, where $\llbracket t \rrbracket_{G_k^a} = \llbracket t \rrbracket_{(k, \bar{G}^a)} = \llbracket (\text{SERVICE } k \ t) \rrbracket_{(j_0, \bar{G}^a)}$.

Suppose that there is a mapping $\nu \in \pi_{var(t)}(\llbracket P \rrbracket_{G^i})$. Therefore, there exists a mapping $\nu_{ext} \in \llbracket P \rrbracket_{G^i}$ such that $\nu \subseteq \nu_{ext}$. This means that $\nu_{ext}(P) \subseteq G^i$. Now, to $\nu_{ext}(P)$, we apply the associated CONSTRUCT query $Q_C = (P_C, P_C \cup P'_C)$ of the completeness statement C that is the witness to generate (SERVICE k t) from the smart set. Note that C has an index k and G_k^a is in \bar{G}^a . This means that there is a mapping μ_C such that $\mu_C(P_C \cup P'_C) \subseteq \tilde{P}$ and $\tilde{t} \in \mu_C(P_C)$. Thus, it follows that

- $\nu_{ext} \tilde{id}^{-1} \mu_C(P_C \cup P'_C) \subseteq \nu_{ext} \tilde{id}^{-1}(\tilde{P}) = \nu_{ext}(P)$ and
- $\nu_{ext} \tilde{id}^{-1}(\tilde{t}) \in \nu_{ext} \tilde{id}^{-1} \mu_C(P_C)$.

We now have that

$$v_{ext} \tilde{id}^{-1}(\tilde{t}) \in v_{ext} \tilde{id}^{-1} \mu_C(P_C) \subseteq \llbracket Q_C \rrbracket_{v_{ext}(P)} \subseteq \llbracket Q_C \rrbracket_{G^i} \subseteq G_k^a,$$

where the last inclusion is due to $(G_k^a, G^i) \models C$ (recall that we assume that $\tilde{\mathcal{G}} \models_{fed} \tilde{C}$ holds). Therefore, it is the case that $v_{ext} \tilde{id}^{-1}(\tilde{t}) \in G_k^a$. Since $\pi_{var(\tilde{id}^{-1}(\tilde{t}))}(v_{ext}) = v$, this implies that $v(t) \in G_k^a$ and, thus, $v \in \llbracket t \rrbracket_{G_k^a}$. As v was arbitrary, it therefore holds that $\pi_{var(t)}(\llbracket P \rrbracket_{G^i}) \subseteq \llbracket t \rrbracket_{G_k^a}$.

Since it holds for every SERVICE pattern (SERVICE k t) occurring in $P_{\tilde{S}}$ and it is also the case that $(P_{\tilde{S}})^{\#} = P$ by construction, if we then join all the mapping results from all the SERVICE patterns, we will get that $\llbracket P \rrbracket_{G^i} \subseteq \llbracket P_{\tilde{S}} \rrbracket_{(j_0, \tilde{G}^a)}$.

By considering that $\llbracket P \rrbracket_{G^i} = \llbracket (P_{\tilde{S}})^{\#} \rrbracket_{G^i} \supseteq \llbracket P_{\tilde{S}} \rrbracket_{(j_0, \tilde{G}^a)}$, where the last relation holds due to the nature of available graphs, it then holds that $\llbracket P \rrbracket_{G^i} = \llbracket (P_{\tilde{S}})^{\#} \rrbracket_{G^i} = \llbracket P_{\tilde{S}} \rrbracket_{(j_0, \tilde{G}^a)}$. This proves that \tilde{Q} is a smart rewriting with regard to Q and \tilde{C} , and $Compl(\tilde{Q})$ is entailed by \tilde{C} . \square

J PROOF OF THEOREM 4.18

THEOREM 4.18. *Let \tilde{C} be a set of indexed completeness statements and let $Q = (W, P)$ be a basic query. Then,*

$$\text{there exists a smart rewriting of } Q \text{ with regard to } \tilde{C} \quad \text{iff} \quad \tilde{C} \models_{fed} Compl(Q).$$

PROOF. “ \Rightarrow ” We will prove the claim by contradiction. Suppose that there exists a smart rewriting $\tilde{Q} = (W, \tilde{P})$ of Q , but $\tilde{C} \not\models_{fed} Compl(Q)$. Consider a witness for the nonentailment, that is, $\tilde{\mathcal{G}} = (\tilde{G}^a, G^i)$ satisfying \tilde{C} , but $\llbracket Q \rrbracket_{\bigcup_{j \in J} G_j^a} \neq \llbracket Q \rrbracket_{G^i}$. In other words, there is a mapping $\mu \in \llbracket P \rrbracket_{G^i}$ such that $\mu \notin \llbracket P \rrbracket_{\bigcup_{j \in J} G_j^a}$. This means that $\mu(P) \subseteq G^i$ but $\mu(P) \not\subseteq \bigcup_{j \in J} G_j^a$. Thus, there must be some triple pattern $t \in P$ such that $\mu(t) \notin \bigcup_{j \in J} G_j^a$. Since \tilde{Q} is a smart rewriting of Q , this implies that $\mu \notin \llbracket \tilde{P} \rrbracket_{(j_0, \tilde{G}^a)}$. This means that in the SERVICE pattern (SERVICE k t) in \tilde{Q} , it is the case that $\pi_{var(t)}(\mu) \notin \llbracket t \rrbracket_{G_k^a}$ or, equivalently, $\mu(t) \notin G_k^a$.

Now, let l be another source index in J . Introduce a new incomplete FDS (\tilde{H}^a, G^i) such that $H_l^a := G^i$, and let the other components in \tilde{H}^a equal to those of \tilde{G}^a . It easily follows that $(\tilde{H}^a, G^i) \models_{fed} \tilde{C}$. However, it is the case that $\mu \in \llbracket P \rrbracket_{\bigcup_{j \in J} H_j^a}$ but $\mu \notin \llbracket \tilde{P} \rrbracket_{(j_0, \tilde{H}^a)}$, because it still holds that $\pi_{var(t)}(\mu) \notin \llbracket t \rrbracket_{H_k^a}$. Thus, $\llbracket P \rrbracket_{\bigcup_{j \in J} H_j^a} \neq \llbracket \tilde{P} \rrbracket_{(j_0, \tilde{H}^a)}$ in contradiction to our assumption that \tilde{Q} is a smart rewriting.

“ \Leftarrow ” It follows immediately from Lemma 4.17. \square

K PROOF OF LEMMA 5.6

LEMMA 5.6 (ENTAILMENT OF QUERY COMPLETENESS AT A DATE). *Let \hat{C} be a set of timestamped completeness statements, let $Q = (W, P)$ be a query, and let d be a date. Then,*

$$\hat{C} \models Compl(Q, d) \quad \text{iff} \quad \tilde{P} \subseteq T_{\hat{C}_{\geq d}}(\tilde{P}).$$

PROOF. “ \Rightarrow ” We prove by contrapositive. We first consider the case in which $d \in \mathbb{N}$. Assume that $\tilde{P} \not\subseteq T_{\hat{C}_{\geq d}}(\tilde{P})$. We show that $\hat{C} \not\models Compl(Q, d)$ by giving a counterexample series \mathcal{S} such that $\mathcal{S} \models \hat{C}$ but $\mathcal{S} \not\models Compl(Q, d)$, which can be constructed as follows:

$$\mathcal{S} = (G_{now}^a, (\emptyset, \dots, \emptyset, G_d^i, G_{d+1}^i, \dots)),$$

where now is any date such that $now \geq \max(date(\hat{C}) \setminus \{\infty\})$, $G_{now}^a = T_{\hat{C}_{\geq d}}(\tilde{P})$, and $G_d^i = G_{d+1}^i = \dots = \tilde{P}$. By construction, we have that $\mathcal{S} \models \hat{C}$. However, by the assumption that $\tilde{P} \not\subseteq T_{\hat{C}_{\geq d}}(\tilde{P})$, it

is the case that $\llbracket Q \rrbracket_{G_d^i} = \llbracket Q \rrbracket_{\tilde{P}} \not\subseteq \llbracket Q \rrbracket_{T_{\hat{C}_{\geq d}}(\tilde{P})} = \llbracket Q \rrbracket_{G_{now}^a}$ because the freeze mapping \tilde{id} in $\llbracket P \rrbracket_{\tilde{P}}$ is missing in $\llbracket P \rrbracket_{T_{\hat{C}_{\geq d}}(\tilde{P})}$. Therefore, $\mathcal{S} \not\models \text{Compl}(Q, d)$.

The proof for the case in which $d = \infty$ can be done analogously. In this case, we take a date $now > \max(\text{date}(\hat{C}) \setminus \{\infty\})$ to show that $\hat{C} \not\models \text{Compl}(Q, d)$.

“ \Leftarrow ” We first prove the case in which $d \in \mathbb{N}$. Assume that $\tilde{P} \subseteq T_{\hat{C}_{\geq d}}(\tilde{P})$. We will show that $\hat{C} \models \text{Compl}(Q, d)$.

Take a series $\mathcal{S} \models \hat{C}$. We have to show that $\mathcal{S} \models \text{Compl}(Q, d)$, that is, $\llbracket Q \rrbracket_{G_d^i} \subseteq \llbracket Q \rrbracket_{G_{now}^a}$. Suppose there is a mapping $\mu \in \llbracket Q \rrbracket_{G_d^i}$. Thus, there must be a mapping $\mu_{ext} \supseteq \mu$, where $\mu_{ext} \in \llbracket P \rrbracket_{G_d^i}$. We will prove that $\mu_{ext} \in \llbracket P \rrbracket_{G_{now}^a}$. By the assumption that $\tilde{P} \subseteq T_{\hat{C}_{\geq d}}(\tilde{P})$ and the prototypicality of \tilde{P} , it holds that $\mu_{ext} \tilde{id}^{-1}(\tilde{P}) \subseteq T_{\hat{C}_{\geq d}}(\mu_{ext} \tilde{id}^{-1}(\tilde{P}))$. The inclusion can be further extended to $\mu_{ext} \tilde{id}^{-1}(\tilde{P}) \subseteq T_{\hat{C}_{\geq d}}(\mu_{ext} \tilde{id}^{-1}(\tilde{P})) \subseteq T_{\hat{C}_{\geq d}}(G_d^i)$, where the last subsumption holds due to $\mu_{ext} \in \llbracket P \rrbracket_{G_d^i}$. By $\mathcal{S} \models \hat{C}$, it must be the case that $T_{\hat{C}_{\geq d}}(G_d^i) \subseteq G_{now}^a$. Therefore, $\mu_{ext} \tilde{id}^{-1}(\tilde{P}) \subseteq G_{now}^a$, which implies that $\mu_{ext} \in \llbracket P \rrbracket_{G_{now}^a}$.

The proof for the case in which $d = \infty$ can be done analogously. In this case, the assumption $\tilde{P} \subseteq T_{\hat{C}_{\geq \infty}}(\tilde{P})$ is used to show that $\hat{C} \models \text{Compl}(Q, d)$ for any date $d \in \mathbb{N}$. \square

L PROOF OF COROLLARY 5.10

COROLLARY 5.10 (COMPLEXITY OF DECIDING THE GUARANTEED COMPLETENESS DATE). *Deciding whether $\text{gcd}(Q, \hat{C}) \geq d$, given a query Q , a set \hat{C} of timestamped completeness statements, and a date d , is NP-complete.*

PROOF. From Theorem 5.8, there exists an NP procedure to check if $\text{gcd}(Q, \hat{C}) \geq d$. It is NP-hard by reduction from the problem of completeness entailment for basic queries. \square

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback.

REFERENCES

- Maribel Acosta, Elena Simperl, Fabian Flöck, and Maria-Esther Vidal. 2017. Enhancing answer completeness of SPARQL queries via crowdsourcing. *Journal of Web Semantics* 45 (2017), 41–62.
- Keith Alexander, Richard Cyganiak, Michael Hausenblas, and Jun Zhao. March 3, 2011. *Describing Linked Datasets with the VoID Vocabulary*. W3C Interest Group Note. Retrieved February 1, 2015 from <http://www.w3.org/TR/2011/NOTE-void-20110303/>.
- Renzo Angles and Claudio Gutierrez. 2008. The expressive power of SPARQL. In *Proceedings of The Semantic Web - 7th International Semantic Web Conference (ISWC'08), Karlsruhe, Germany, October 26-30, 2008*. 114–129.
- Marcelo Arenas, Claudio Gutierrez, and Jorge Pérez. 2010. On the semantics of SPARQL. In *Semantic Web Information Management - A Model-Based Perspective*. Springer.
- Donovan Artz and Yolanda Gil. 2007. A survey of trust in computer science and the semantic web. *Journal of Web Semantics* 5, 2, 58–71. DOI : <http://dx.doi.org/10.1016/j.websem.2007.03.002>
- Margherita Barile. 2016. Family - Wolfram MathWorld. Retrieved May 13, 2018 from <http://mathworld.wolfram.com/Family.html>.
- Carlo Batini and Monica Scannapieco. 2016. *Data and Information Quality - Dimensions, Principles and Techniques*. Springer.
- David Becker, Trish Dunn King, and Bill McMullen. 2015. Big data, big data quality problem. In *2015 IEEE International Conference on Big Data (Big Data'15), Santa Clara, CA, USA, October 29 - November 1, 2015*. 2644–2653. DOI : <http://dx.doi.org/10.1109/BigData.2015.7364064>
- Wouter Beek, Laurens Rietveld, Hamid R. Bazoobandi, Jan Wielemaker, and Stefan Schlobach. 2014. LOD Laundromat: A uniform way of publishing other people’s dirty data. In *Proceedings of The Semantic Web - 13th International Semantic Web Conference (ISWC'14), Riva del Garda, Italy, October 19-23, 2014. Part I*. 213–228. DOI : http://dx.doi.org/10.1007/978-3-319-11964-9_14

- Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. 2009. DBpedia – A crystallization point for the web of data. *Journal of Web Semantics* 7, 3 (2009), 154–165.
- Dan Brickley and Ramanathan V. Guha (Eds.). 2004. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation. Retrieved Feb 1, 2015 from <http://www.w3.org/TR/2014/REC-rdf-schema-20140225/>.
- Carlos Buil-Aranda, Marcelo Arenas, Óscar Corcho, and Axel Polleres. 2013. Federating queries in SPARQL 1.1: Syntax, semantics and evaluation. *Journal of Web Semantics* 18, 1, 1–17. DOI: <http://dx.doi.org/10.1016/j.websem.2012.10.001>
- Ashok K. Chandra and Philip M. Merlin. 1977. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the 9th ACM Symposium on Theory of Computing (STOC'77)*.
- Surajit Chaudhuri and Moshe Y. Vardi. 1993. Optimization of real conjunctive queries. In *Proceedings of the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'93)*.
- E. F. Codd. 1979. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems* 4, 4, 397–434. DOI: <http://dx.doi.org/10.1145/320107.320109>
- Richard Cyganiak, David Wood, and Markus Lanthaler (Eds.). 2014. *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation. Retrieved January 15, 2017 from <https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/>.
- Fariz Darari, Werner Nutt, Giuseppe Pirrò, and Simon Razniewski. 2013. Completeness statements about RDF data sources and their use for query answering. In *Proceedings of the 12th International Semantic Web Conference (ISWC'13)*.
- Fariz Darari, Werner Nutt, Giuseppe Pirrò, and Simon Razniewski. 2016. Toward efficient techniques for completeness reasoning. In *Technical Report*. Retrieved May 13, 2018 from <http://completeness.inf.unibz.it/TR-2016-toward-efficient-completeness-reasoning.pdf>.
- Fariz Darari, Radityo Eko Prasoj, and Werner Nutt. 2014. CORNER: A completeness reasoner for SPARQL queries over RDF data sources. In *Proceedings of the 11th Extended Semantic Web Conference (ESWC'14) Posters and Demos*.
- Fariz Darari, Radityo Eko Prasoj, and Werner Nutt. 2015. Expressing no-value information in RDF. In *Proceedings of the ISWC'15 Posters & Demonstrations Track co-located with the 14th International Semantic Web Conference (ISWC'15), Bethlehem, PA, USA, October 11, 2015*.
- Fariz Darari, Radityo Eko Prasoj, Simon Razniewski, and Werner Nutt. 2017. COOL-WD: A completeness tool for wikidata. In *Proceedings of the ISWC'17 Posters & Demonstrations and Industry Tracks co-located with 16th International Semantic Web Conference (ISWC'17), Vienna, Austria, October 23-25, 2017*. <http://ceur-ws.org/Vol-1963/paper466.pdf>.
- Fariz Darari, Simon Razniewski, Radityo Eko Prasoj, and Werner Nutt. 2016. Enabling fine-grained RDF data completeness assessment. In *Proceedings of Web Engineering - 16th International Conference (ICWE 2'16), Lugano, Switzerland, June 6-9, 2016*. 170–187.
- DCMI (Ed.). 14 June 2012. *DCMI Metadata Terms*. DCMI Recommendation. Retrieved December 5, 2017 from <http://dublincore.org/documents/2012/06/14/dcmi-terms/>.
- AnHai Doan, Alon Halevy, and Zachary Ives. 2012. *Principles of Data Integration*. Morgan Kaufmann, Burlington, MA.
- Fredo Erxleben, Michael Günther, Markus Krötzsch, Julian Mendez, and Denny Vrandečić. 2014. Introducing Wikidata to the linked data web. In *Proceedings of The Semantic Web - 13th International Semantic Web Conference (ISWC'14), Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*. 50–65.
- Jérôme Euzenat and Pavel Shvaiko. 2013. *Ontology Matching* (2nd ed.). Springer-Verlag, Heidelberg (DE). <http://book.ontologymatching.org>.
- Valeria Fionda, Giuseppe Pirrò, and Mariano P. Consens. 2015a. Extended property paths: Writing more SPARQL queries in a succinct way. In *29th AAAI Conference on Artificial Intelligence*. 102–108.
- Valeria Fionda, Giuseppe Pirrò, and Claudio Gutierrez. 2015b. NautiLOD: A formal language for the web of data graph. *ACM Transactions on the Web* 9, 1 (2015), 5:1–5:43.
- Christian Fürber and Martin Hepp. 2010. Using SPARQL and SPIN for data quality management on the semantic web. In *Proceedings of the 13th International Conference on Business Information Systems (BIS'10)*.
- Christian Fürber and Martin Hepp. 2011. Towards a vocabulary for data quality management in semantic web architectures. In *Proceedings of the 2011 EDBT/ICDT Workshop on Linked Web Data Management*.
- Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. 2013. AMIE: Association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22nd International Conference on World Wide Web (WWW'13)*.
- Claudio Gutiérrez, Carlos A. Hurtado, and Alejandro A. Vaisman. 2005. Temporal RDF. In *Proceedings of The Semantic Web: Research and Applications, 2nd European Semantic Web Conference (ESWC'05), Heraklion, Crete, Greece, May 29 - June 1, 2005*. 93–107.
- Steve Harris and Andy Seaborne (Eds.). 2013. *SPARQL 1.1 Query Language*. W3C Recommendation. Retrieved February 1, 2015 from <http://www.w3.org/TR/2013/REC-sparql11-query-20130321/>.
- Andreas Harth and Sebastian Speiser. 2012. On completeness classes for query evaluation on linked data. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI'12)*.
- Olaf Hartig. 2009. Provenance information in the web of data. In *Proceedings of the WWW'09 Workshop on Linked Data on the Web (LDOW'09), Madrid, Spain, April 20, 2009*. http://ceur-ws.org/Vol-538/ldow2009_paper18.pdf.

- Olaf Hartig, Christian Bizer, and Johann-Christoph Freytag. 2009. Executing SPARQL queries over the web of linked data. In *Proceedings of the 8th International Semantic Web Conference (ISWC'09)*.
- Olaf Hartig and Giuseppe Pirrò. 2015. A context-based semantics for SPARQL property paths over the web. In *European Semantic Web Conference*. Springer, 71–87.
- Olaf Hartig and Giuseppe Pirrò. 2017. SPARQL with property paths on the web. *Semantic Web* 8, 6, 773–795.
- Oktie Hassanzadeh and Mariano P. Consens. 2009. Linked movie data base. In *Proceedings of the WWW'09 Workshop on Linked Data on the Web (LDOW'09), Madrid, Spain, April 20, 2009*.
- Patrick J. Hayes and Peter F. Patel-Schneider (Eds.). 2014. *RDF 1.1 Semantics*. W3C Recommendation. Retrieved May 27, 2016 from <https://www.w3.org/TR/2014/REC-rdf11-mt-20140225/>.
- Tom Heath and Christian Bizer. 2011. *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool, Burlington, MA.
- Sven Helmer and Guido Moerkotte. 2003. A performance study of four index structures for set-valued attributes of low cardinality. *VLDB Journal* 12, 3 (2003), 244–261.
- David Hilbert. 1902. Mathematical problems. *Bulletin of the American Mathematical Society* 8, 10, 437–479.
- Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph (Eds.). 2012. *OWL 2 Web Ontology Language Primer (Second Edition)*. W3C Recommendation. Retrieved February 1, 2015 from <http://www.w3.org/TR/2012/REC-owl2-primer-20121211/>.
- Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, Edwin Lewis-Kelham, Gerard de Melo, and Gerhard Weikum. 2011. YAGO2: Exploring and querying world knowledge in time, space, context, and many languages. In *Proceedings of the 20th International Conference on World Wide Web (WWW'11)*.
- Jörg Hoffmann and Jana Koehler. 1999. A new method to index and query sets. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*.
- Aidan Hogan. 2015. Skolemising blank nodes while preserving isomorphism. In *Proceedings of the 24th International Conference on World Wide Web (WWW'15), Florence, Italy, May 18-22, 2015*. 430–440.
- Aidan Hogan, Marcelo Arenas, Alejandro Mallea, and Axel Polleres. 2014. Everything you always wanted to know about blank nodes. *Journal of Web Semantics* 27, 42–69.
- Mark Kaminski and Egor V. Kostylev. 2016. Beyond well-designed SPARQL. In *19th International Conference on Database Theory (ICDT'16), Bordeaux, France, March 15-18, 2016*. 5:1–5:18. DOI : <http://dx.doi.org/10.4230/LIPIcs.ICDT.2016.5>
- Graham Klyne and Jeremy J. Carroll (Eds.). 2004. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation. Retrieved February 1, 2015 from <http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>.
- Holger Knublauch and Dimitris Kontokostas (Eds.). 2017. *Shapes Constraint Language (SHACL)*. W3C Candidate Recommendation. Retrieved May 20, 2017 from <https://www.w3.org/TR/2017/CR-shacl-20170411/>.
- Timothy Lebo, Satya Sahoo, and Deborah McGuinness (Eds.). 2012. *PROV-O: The PROV Ontology*. W3C Candidate Recommendation. Retrieved May 27, 2016 from <https://www.w3.org/TR/2012/CR-prov-o-20121211/>.
- Andrés Letelier, Jorge Pérez, Reinhard Pichler, and Sebastian Skritek. 2012. Static analysis and optimization of semantic web queries. In *Proceedings of the 31st Symposium on Principles of Database Systems (PODS'12)*.
- Alon Y. Levy. 1996. Obtaining complete answers from incomplete databases. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB'96)*.
- Antoni Ligeza. 2006. *Logical Foundations for Rule-Based Systems*, 2nd Ed. Studies in Computational Intelligence, Vol. 11. Springer.
- Nuno Lopes, Axel Polleres, Umberto Straccia, and Antoine Zimmermann. 2010. AnQL: SPARQLing up annotated RDFS. In *The Semantic Web - 9th International Semantic Web Conference (ISWC'10), Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I*. 518–533.
- Carsten Lutz, Inanç Seylan, and Frank Wolter. 2013. Ontology-based data access with closed predicates is inherently intractable (sometimes). In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13), Beijing, China, August 3-9, 2013*. 1024–1030. <http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6870>.
- Carsten Lutz, Inanç Seylan, and Frank Wolter. 2015. Ontology-mediated queries with closed predicates. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15), Buenos Aires, Argentina, July 25-31, 2015*. 3120–3126. <http://ijcai.org/Abstract/15/440>.
- Fadi Maali and John Erickson. 2014. *Data Catalog Vocabulary (DCAT)*. W3C Recommendation. Retrieved December 5, 2017 from <http://www.w3.org/TR/2014/REC-vocab-dcat-20140116/>.
- Frank Manola and Eric Miller (Eds.). 2004. *RDF Primer*. W3C Recommendation. Retrieved July 31, 2016 from <https://www.w3.org/TR/2004/REC-rdf-primer-20040210/>.
- Pablo N. Mendes, Hannes Mühleisen, and Christian Bizer. 2012. Sieve: Linked data quality assessment and fusion. In *Proceedings of the 2012 EDBT/ICDT Workshop on Linked Web Data Management*.
- Paramita Mirza, Simon Razniewski, Fariz Darari, and Gerhard Weikum. 2017. Cardinal virtues: Extracting relation cardinalities from text. In *ACL'17 Short Papers*.

- Paramita Mirza, Simon Razniewski, and Werner Nutt. 2016. Expanding Wikidata's parenthood information by 178%, or how to mine relation cardinalities. In *ISWC Posters & Demos*.
- Amihai Motro. 1989. Integrity = Validity + completeness. *ACM Transactions on Database Systems* 14, 4 (1989), 480–502.
- Sergio Muñoz, Jorge Pérez, and Claudio Gutierrez. 2009. Simple and efficient minimal RDFS. *Journal of Web Semantics* 7, 3 (2009), 220–234.
- Nhung Ngo, Magdalena Ortiz, and Mantas Simkus. 2016. Closed predicates in description logics: Results on combined complexity. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 15th International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016*. 237–246. <http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12906>.
- Natasha Noy and Alan Rector (Eds.). 2006. *Defining N-ary Relations on the Semantic Web*. W3C Working Group Note. Retrieved January 10, 2017 from <https://www.w3.org/TR/2006/NOTE-swbp-n-aryRelations-20060412/>.
- Peter F. Patel-Schneider. 2015. Using description logics for RDF constraint checking and closed-world recognition. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*. AAAI Press, 247–253. <http://dl.acm.org/citation.cfm?id=2887007.2887042>.
- Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. 2009. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems* 34, 3 (2009), 16:1–16:45.
- François Picalausa and Stijn Vansummeren. 2011. What are real SPARQL queries like? In *Proceedings of the International Workshop on Semantic Web Information Management (SWIM'11)*.
- Radityo Eko Prasajo, Fariz Darari, Simon Razniewski, and Werner Nutt. 2016. Managing and consuming completeness information for Wikidata using COOL-WD. In *Proceedings of the 7th International Workshop on Consuming Linked Data co-located with 15th International Semantic Web Conference (COLID@ISWC'15), Kobe, Japan, October 18, 2016*. <http://ceur-ws.org/Vol-1666/paper-02.pdf>.
- Eric Prud'hommeaux, Iovka Boneva, Jose Emilio Labra Gayo, and Gregg Kellogg (Eds.). 2017. *Shape Expressions Language 2.0*. W3C Community Group Draft Report. Retrieved December 12, 2017 from <http://shex.io/shex-semantics-20170713>.
- Eric Prud'hommeaux and Carlos Buil-Aranda (Eds.). *SPARQL 1.1 Federated Query*. W3C Recommendation. Retrieved February 1, 2015 from <http://www.w3.org/TR/2013/REC-sparql11-federated-query-20130321/>.
- Andrea Pugliese, Octavian Udrea, and V. S. Subrahmanian. 2008. Scaling RDF with time. In *Proceedings of the 17th International Conference on World Wide Web (WWW'08), Beijing, China, April 21-25, 2008*. 605–614.
- Simon Razniewski and Werner Nutt. 2011. Completeness of queries over incomplete databases. In *Proceedings of the 37th International Conference on Very Large Data Bases (VLDB'11)*.
- Muhammad Saleem, Muhammad Intizar Ali, Aidan Hogan, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo. 2015. LSQ: The linked SPARQL queries dataset. In *Proceedings of The Semantic Web - 14th International Semantic Web Conference (ISWC'15), Bethlehem, PA, USA, October 11-15, 2015, Part II*. 261–269. DOI : http://dx.doi.org/10.1007/978-3-319-25010-6_15
- Iztok Sarnik. 2013. Index data structure for fast subset and superset queries. In *International Cross Domain Conference and Workshop (CD-ARES'13)*.
- Max Schmachtenberg, Christian Bizer, and Heiko Paulheim. 2014. Adoption of the linked data best practices in different topical domains. In *Proceedings of The Semantic Web - 13th International Semantic Web Conference (ISWC'14), Riva del Garda, Italy, October 19-23, 2014. Part I*. 245–260.
- Michael Schmidt, Michael Meier, and Georg Lausen. 2010. Foundations of SPARQL query optimization. In *Proceedings of Database Theory - 13th International Conference (ICDT'10), Lausanne, Switzerland, March 23-25, 2010*, 4–33. DOI : <http://dx.doi.org/10.1145/1804669.1804675>
- Inanç Seylan, Enrico Franconi, and Jos de Bruijn. 2009. Effective query rewriting with ontologies over DBoxes. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09), Pasadena, California, USA, July 11-17, 2009*. 923–925. <http://ijcai.org/Proceedings/09/Papers/157.pdf>.
- Giorgos Stoilos, Bernardo Cuenca Grau, and Ian Horrocks. 2010. How incomplete is your semantic web reasoner? In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI'10)*.
- Jonas Tappolet and Abraham Bernstein. 2009. Applied temporal RDF: Efficient temporal querying of RDF data with SPARQL. In *Proceedings of The Semantic Web: Research and Applications, 6th European Semantic Web Conference (ESWC'09), Heraklion, Crete, Greece, May 31-June 4, 2009*. 308–322.
- Richard Y. Wang and Diane M. Strong. 1996. Beyond accuracy: What data quality means to data consumers. *Journal of Management Information Systems* 12, 4 (1996), 5–33.
- Xin Wang, Howard J. Hamilton, and Yashu Bither. 2005. *An Ontology-based Approach to Data Cleaning*. Technical Report. Department of Computer Science, University of Regina.
- Amrapali Zaveri, Anisa Rula, Andrea Maurino, Ricardo Pietrobon, Jens Lehmann, and Sören Auer. 2016. Quality assessment for linked data: A survey. *Semantic Web* 7, 1, 63–93.

Received August 2016; revised February 2018; accepted February 2018