Studies on the Semantic Web

Fariz Darari

Managing and Consuming Completeness Information for RDF Data Sources

IOS Press

AKA

Studies on the Semantic Web

ISSN 1868-1158 (print)
ISSN 2215-0870 (online)

SSW 042

Managing and Consuming Completeness Information for RDF Data Sources

The increasing amount of structured data available on the Web is laying the foundations for a global-scale knowledge base. But the ever increasing amount of Semantic Web data gives rise to the question – how complete is that data? Though data on the Semantic Web is generally incomplete, some may indeed be complete.

In this book, the author deals with how to manage and consume completeness information about Semantic Web data. In particular, the book explores how completeness information can guarantee the completeness of query answering. Optimization techniques for completeness reasoning and the conducting of experimental evaluations are provided to show the feasibility of the approaches, as well as a technique for checking the soundness of queries with negation via reduction to query completeness checking.

Other topics covered include completeness information with timestamps, and two demonstrators – CORNER and COOL-WD – are provided to show how a completeness framework can be realized. Finally, the book investigates an automated method to generate completeness statements from text on the Web.

The book will be of interest to anyone whose work involves dealing with Web-data completeness.

9 781643 680347

IOS Press
www.iospress.nl

AKA
www.aka-verlag.com

9 783898 387484

ISBN 978-1-64368-034-7 (IOS Press, print)
ISBN 978-1-64368-035-4 (IOS Press, online)

ISBN 978-3-89838-748-4 (AKA, print)

MANAGING AND CONSUMING COMPLETENESS INFORMATION FOR RDF DATA SOURCES

Studies on the Semantic Web

Volume 042

ISSN 1868-1158 (print)
ISSN 2215-0870 (online)

MANAGING AND CONSUMING COMPLETENESS INFORMATION FOR RDF DATA SOURCES

Fariz Darari

Faculty of Computer Science, Kampus UI Depok, West Java, Indonesia

IOS Press

AKA

ISBN 978-3-89838-748-4 (AKA, print)
ISBN 978-1-64368-034-7 (IOS Press, print)
ISBN 978-1-64368-035-4 (IOS Press, online)
doi: 10.3233/SSW42

Bibliographic information available from the Katalog der Deutschen Nationalbibliothek (German National Library Catalogue) at https://www.dnb.de

Dissertation, approved by the Free University of Bozen-Bolzano, Italy and Technische Universität Dresden, Germany
Date of the defense: 14 July 2017
Supervisors: Prof. Werner Nutt, Prof. Sebastian Rudolph
Exam Committee Members: Prof. Franz Baader, Prof. Carlo Combi, Prof. Markus Zanker

Publisher
Akademische Verlagsgesellschaft AKA GmbH, Berlin

Represented by Co-Publisher IOS Press
IOS Press BV
Nieuwe Hemweg 6B
1013 BG Amsterdam
The Netherlands
Tel: +31 20 688 3355
Fax: +31 20 687 0019
email: order@iospress.nl

Abstract

The ever increasing amount of Semantic Web data gives rise to the question: How complete is the data? Though generally data on the Semantic Web is incomplete, many parts of data are indeed complete, such as the children of Barack Obama and the crew of Apollo 11. This thesis aims to study how to manage and consume completeness information about Semantic Web data. In particular, we first discuss how completeness information can guarantee the completeness of query answering. Next, we propose optimization techniques of completeness reasoning and conduct experimental evaluations to show the feasibility of our approaches. We also provide a technique to check the soundness of queries with negation via reduction to query completeness checking. We further enrich completeness information with timestamps, enabling query answers to be checked up to when they are complete. We then introduce two demonstrators, i.e., CORNER and COOL-WD, to show how our completeness framework can be realized. Finally, we investigate an automated method to generate completeness statements from text on the Web via relation cardinality extraction.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The increasing amount of structured data made available on the Web is laying the foundation of a global-scale knowledge base. Projects like Linked Open Data (LOD) [50], by inheriting some basic design principles of the Web (e.g, simplicity, decentralization), aim at making huge volumes of data available via the Resource Description Framework (RDF) standard data model [59]. RDF enables one to make *statements* about resources in the form of triples, consisting of a *subject*, a *predicate*, and an *object*. The common path to access such a huge amount of structured data is via SPARQL endpoints, namely, network locations that can be queried using the SPARQL query language [46].

With a large number of RDF data sources (i.e., 1139 data sources in 2017 as recorded by the LOD Cloud[1]), covering possibly overlapping knowledge domains, it is natural to observe a wide range of data source quality. Indeed, depending on the topics and aspects considered, RDF data sources such as Wikidata [111], DBpedia [14], and YAGO [53], may possess different quality characteristics. 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, analytics, and so forth. Proposals like the `VoID` vocabulary [6] touch this aspect. With `VoID` it is possible, among other things, to provide information about the number of instances of a particular class, the

[1]`http://lod-cloud.net/`

SPARQL endpoint of a source, and links to other data sources. However, `VoID` focuses on providing quantitative information. We claim that toward comprehensive descriptions of data sources, also *qualitative* information is crucial; hence, the overall aim of this thesis is to study a specific aspect of data quality for RDF data sources, that is, *completeness.*

1.1. Data Completeness

Information about completeness is crucial for RDF data sources, where each data source is generally considered incomplete due to the open-world assumption (OWA) [49]. 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 IMDb[2] is *complete for all movies directed by Tarantino.* The possibility to provide in a declarative and machine-readable way such kind of completeness statements paves the way toward a new generation of services for consuming data. In this respect, the semantics of completeness statements interpreted by a reasoning engine can, for instance, guarantee the completeness of query answers.

Data completeness, as defined by Wang and Strong [112], is the *breadth*, *depth*, and *scope* of information contained in the data. Batini and Scannapieco [12] considered data completeness to be one of the most significant data quality dimensions. Like other quality dimensions (e.g., accuracy, timeliness), the problem of data completeness may occur in various application domains, such as biology, aviation, and healthcare, as studied by Becker et al. [13].

Concerns about data (in-)completeness in the field of relational databases, can be traced back to 1979 [22], where Codd proposed a treatment of nulls based on three-valued logic. Motro [80] developed an integrity model for databases that considers completeness (and validity). Levy [64] 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 [96] reduced the problem of query completeness

[2]`http://www.imdb.com/`

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 due to the OWA. Several researchers studied completeness in the broader context of data quality. Fürber and Hepp [38] 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. [73] 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.[3] The group is developing SHACL, a language for validating RDF graphs against a set of conditions (called 'shapes') [60]. In SHACL, one can formulate integrity constraints, e.g., 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, like a spouse, is missing.

Zaveri et al. [114] surveyed techniques to measure the completeness (among other data quality aspects) of RDF data sources. It is common to these techniques that they 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 did not concern how to express that a source is of gold-standard quality for some type of information. In [47], Harth and Speiser 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. [41] stressed the need of complete information for rule mining

[3]`https://www.w3.org/2014/data-shapes`

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.

In RDF, an existing way to state completeness is by using closed lists (called 'RDF collections') [71]. Such lists, however, introduce a new structure, that is different from the usual SPO-style of RDF triples, hence hindering data access via querying. In description logics (DLs), several proposals have been made for partial closed-world features. OWL (i.e., the DL-based ontology language for the Semantic Web) provides a functionality to describe a closed class by enumerating all of its instances [52]. Seylan et al. [105] introduced DBoxes to capture DB-style relations for DL ontologies. They developed procedures to translate implicitly defined queries over DBoxes into explicitly defined ones. A similar approach was proposed by Lutz et al. [67, 68] with their closed predicates that allow one to draw more conclusions in DL reasoning. However, they showed that this also leads to higher complexity of the reasoning. Ahmetaj et al. [5] observed that a query over a DL ontology can have more certain answers if some predicates are assumed to be closed. They showed how to rewrite a simple kind of queries, so-called instance queries, that ask for all instances of a class or a property into a datalog query, so that the rewriting retrieves all the certain answers of the original query. Ngo et al. [83] showed how closed predicates increase the combined complexity even for simple queries in some well-studied DL dialects. The problem of checking query completeness was not considered in the above work, as they were only interested in drawing more conclusions of reasoning with closed predicates.

Among the first proposals for a declarative, machine-readable specification of Semantic Web data completeness was the work by Darari et al. [27], which enables us to close some parts of RDF data, and thus SPARQL queries can be answered completely whenever they touch only the closed parts. The impact of completeness statements on a variety of SPARQL fragments, including the RDFS entailment regime and the federated scenario, was studied. The reasoning technique they developed is, however, agnostic of the content of RDF data sources, that is, the query completeness checking considers only the completeness statements, and the specifics of the graph to which the statements are given do not play any role.

Research Hypotheses. As discussed above, previous approaches dealt with limited settings of data completeness for RDF data sources. This thesis aims to develop a comprehensive framework of managing and consuming completeness information for RDF data sources. The hypotheses of this thesis are as follows:

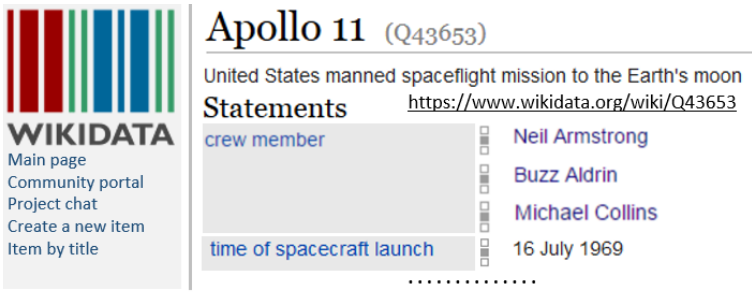
- Combining information about data completeness and the actual data gives rise to a stronger and more fine-grained assessment of the completeness of query answers.
- By applying and adapting existing indexing techniques, query completeness analysis can be performed in a time that is comparable to the execution time of a query.
- Completeness analysis can be leveraged to check whether answers to queries with negation are sound.
- Completeness statements can be equipped with temporal information in such a way that temporal completeness analysis can be performed with little additional cost.
- Existing Semantic Web technologies can be used to develop completeness management tools with little development overhead.
- Natural language texts contain information about cardinalities of sets in the real world that can be extracted automatically and be used to assess the completeness of RDF data sources.

1.2. Motivation

We provide motivating scenarios covering a range of aspects of completeness for RDF data sources: data-aware completeness reasoning, optimizations of completeness reasoning, ensuring query soundness using completeness statements, time-aware completeness reasoning, demonstrators of systems to create and consume completeness statements, and extracting relation cardinalities from text as a way to automatically generate completeness statements.

Data-aware Completeness Reasoning. Consider Wikidata, a crowd-sourced KB with RDF support [111]. For data about the movie Reservoir Dogs, Wikidata is incomplete, as it is missing the fact that

Michael Sottile was acting in the movie.[4] On the other hand, for data about Apollo 11, it is the case that Neil Armstrong, Buzz Aldrin, and Michael Collins, recorded as crew members on Wikidata, are indeed *all* the crew (see Figure 1.1).[5] However, such completeness information is not recorded and thus it is left to the reader to decide whether some data on the Web is already complete.



Apollo 11 (Q43653)

United States manned spaceflight mission to the Earth's moon

Statements https://www.wikidata.org/wiki/Q43653

crew member	Neil Armstrong
	Buzz Aldrin
	Michael Collins
time of spacecraft launch	16 July 1969

....

Figure 1.1. Wikidata is actually complete for all the Apollo 11 crew

Nevertheless, the availability of explicit completeness information can benefit data access over RDF data sources, commonly done via SPARQL queries. For example, suppose that in addition to the complete data of the Apollo 11 crew, Wikidata is also complete for the children of Neil Armstrong, Buzz Aldrin, and Michael Collins. Consequently, a user asking the query "children of Apollo 11 crew" should obtain not only query answers, but also *the information that the query can be answered completely*. Observe that here data-specific reasoning is employed: we first obtain who specifically are the complete crew members of Apollo 11, and then for each of them, we check if we have all the children.

Motivated by the above rationales, we argue that it is important to describe the (partial) completeness of RDF data sources and provide a technique to check query completeness based on RDF data sources with completeness information. We call such a check *completeness entailment*. In previous work, Darari et al. [27] proposed a framework to provide completeness statements about RDF data sources and check query completeness based on such statements. There is, however,

[4]By comparing the data at `https://www.wikidata.org/wiki/Q72962` (as of Sep 18, 2016) with the complete information at `http://www.imdb.com/title/tt0105236/fullcredits`

[5]`http://www.space.com/16758-apollo-11-first-moon-landing.html`

one fundamental limitation of the work: the completeness check is *agnostic* of the content of the RDF data sources to which completeness statements are given, which results in weaker inferences. For instance, given the completeness information and the query "children of Apollo 11 crew" as in the Apollo 11 example above, the data-agnostic approach fails to capture the query completeness. In Chapter 3, we provide a formalization, and a sound and complete algorithm of *data-aware* completeness checking. Moreover, we identify two fragments of completeness statements: *SP-statements*, that are practically relevant to entity-centric, crowdsourced RDF data sources like Wikidata, and *no-value statements*, that are suited to capturing the non-existence of information in RDF.

Optimization Techniques of Completeness Reasoning. Real-world RDF data sources may contain a large amount of data. For example, from the English Wikipedia, DBpedia extracted 580 million RDF triples.[6] Obviously, neither is all information from those triples complete, nor is its completeness interesting. If 20% of those triples were captured by completeness statements, where each statement accounts for 100 triples, then there would be about 1 million statements in total needed for DBpedia.

Now, the question is, how fast can we perform completeness reasoning with 1 million statements? Using a plain completeness reasoner that employs all the completeness statements, we observed that reasoning time may take minutes. Obviously, this is not feasible as we expect that in practice completeness reasoning would be performed as often as query evaluation. Indeed, the reason why a plain reasoner may take long is that it takes into account all the statements in the reasoning. Yet, not all statements contribute to the entailment of query completeness. For instance, the completeness statement "all football players of Arsenal" does not contribute to the completeness of the query "movies directed by Tarantino."

In Chapter 4, we analyze the complexity of the completeness reasoning task in practical settings and propose a *relevance principle*, which allows us to reduce the number of statements considered in the reasoning. Based on the relevance principle, we then develop retrieval techniques of relevant statements with various index structures, and

[6] `http://lists.w3.org/Archives/Public/public-lod/2014Sep/0028.html`

conduct experimental evaluations to study the characteristics of those index structures. Next, we experimentally evaluate completeness reasoning over a realistic setting based on SPARQL query logs of several real-world RDF data sources, i.e., DBpedia, Semantic Web Dog Food (SWDF), and LinkedGeoData (LGD).

Wrt. data-aware completeness reasoning, based on our observation that natural-language completeness statements on the Web are generally about similar topics (e.g., completeness statements about cast of movies on IMDb[7] and about points of interest of cities on OpenStreetMap[8]), we introduce *completeness templates*. Such templates provide a compact representation of similar completeness statements, enabling multiple completeness statements to be processed simultaneously in the reasoning. We then evaluate the performance of data-aware completeness reasoning using completeness templates, over a Wikidata-based experimental setup.

Ensuring Query Soundness Using Completeness Statements. The use of negation in SPARQL has always been problematic. RDF generally follows the open-world assumption (OWA): information recorded in an RDF dataset can be incomplete, that is, it might not reflect all information valid in reality [49]. Consequently, SPARQL queries with negation (which rely on the absence of some information) cannot be guaranteed to deliver sound answers.

To illustrate this, consider asking for "countries that are not EU founders" over the Wikidata SPARQL endpoint:[9,10]

```
SELECT * WHERE {
  ?c wdt:P31 wd:Q6256 . # ?c a (= instanceof) country
  FILTER NOT EXISTS {wd:Q458 wdt:P112 ?c} # EU founder ?c
}
```

The answers include Spain (= `wd:Q29`).[11] We might wonder if this answer is sound, that is, if Spain is indeed a country that is not an EU founder. Without any completeness information about Wikidata,

[7]For instance, on the Reservoir Dogs page at `http://www.imdb.com/title/tt0105236/fullcredits`

[8]For instance, on the Abingdon page at `http://wiki.openstreetmap.org/wiki/Abingdon`

[9]`https://query.wikidata.org/`

[10]Prefix declarations are provided in Appendix A.

[11]Wikidata uses internal identifiers for resources, as also shown in the SPARQL query example.

we cannot be sure about this: assume Spain were indeed a founder of the EU, but this information were missing from the data. Obviously, in that case, Spain is not a correct answer to the above query. In reality, the EU founders are *exactly* the countries Belgium, Germany, France, Italy, Luxembourg, and the Netherlands.[12] Knowing this completeness information guarantees that Spain is a country that is *not* an EU founder.

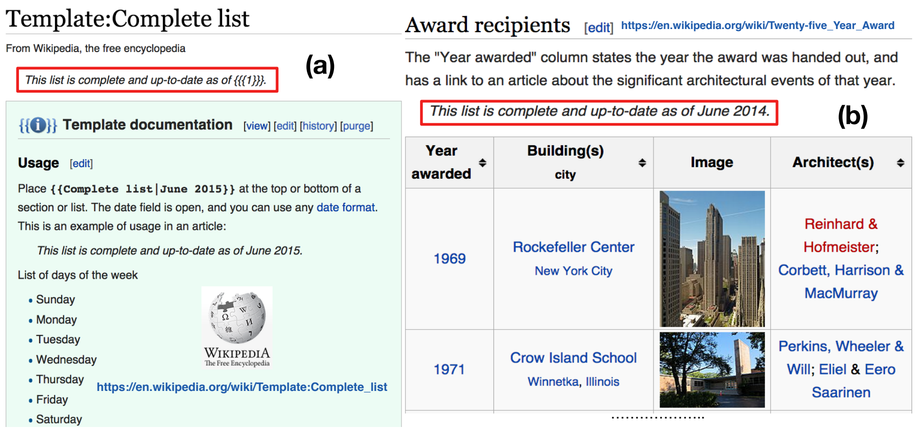
What we can observe here is that, without completeness information, negation in SPARQL may lead to the problem of unsound answers. This is due to the inherent *non-monotonicity* of answering queries with negation: adding new information may invalidate an answer.[13] Having completeness information may then help ensure the *soundness* of answers, that is, we can be sure that specific answers will still be returned, even if the data is completed. Chapter 5 describes our formalization of the problem of query soundness in the presence of completeness statements. We distinguish between the soundness of a specific answer of a graph pattern and the soundness of a graph pattern as a whole. We further provide a characterization of the problem via reduction to completeness checking. Finally, we perform an experimental evaluation of soundness checking in a realistic setup based on Wikidata.

Time-aware Completeness Reasoning. The notion of completeness introduced in [27] is in a sense time-agnostic. It only allows one to specify whether (a portion of) a data source is complete. However, one may also be interested in having completeness guarantees up to a certain time. To cope with this aspect we introduce *timestamped completeness statements*. In defining such kind of statements, we were inspired by Wikipedia which provides a template list for complete information with timestamps, as shown in Figure 1.2.

Figure 1.2 shows a list template taken from Wikipedia. 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 list template 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

[12]`https://europa.eu/european-union/about-eu/history/`

[13]Note that for the positive fragment of SPARQL, such a problem can never occur, as the answers are always sound, thanks to monotonicity.



Template:Complete list

From Wikipedia, the free encyclopedia

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

{{i}} **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
- Friday
- Saturday

WIKIPEDIA The Free Encyclopedia

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

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.

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; Eliel & Eero Saarinen

....

Figure 1.2. 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)

won the Twenty-five Year Award[14] (as shown in Figure 1.2) to the page of Italian DOP cheeses.[15]

In Chapter 6, we provide a formalization of time-aware completeness reasoning. Completeness statements now feature timestamps. Consequently, query completeness must be approached differently. For this reason, we introduce the *guaranteed completeness date* of a query, that is, the latest date for which complete query results are guaranteed to be contained in the actual query results. We then develop, given a set of timestamped completeness statements, an algorithm to compute the guaranteed completeness date of a query, which is optimal in the sense that each timestamped completeness statement is considered at most once in the reasoning.

Completeness Management Demonstrators. The theoretical foundations of completeness reasoning [27] so far have not reached practice. Up to now, users can only write completeness information manually in RDF, and would need to publish and link them on their own, in order to make them available. Similarly, users interested in making use of completeness statements have no central reference for retrieving such information. We believe that the lack of systems supporting

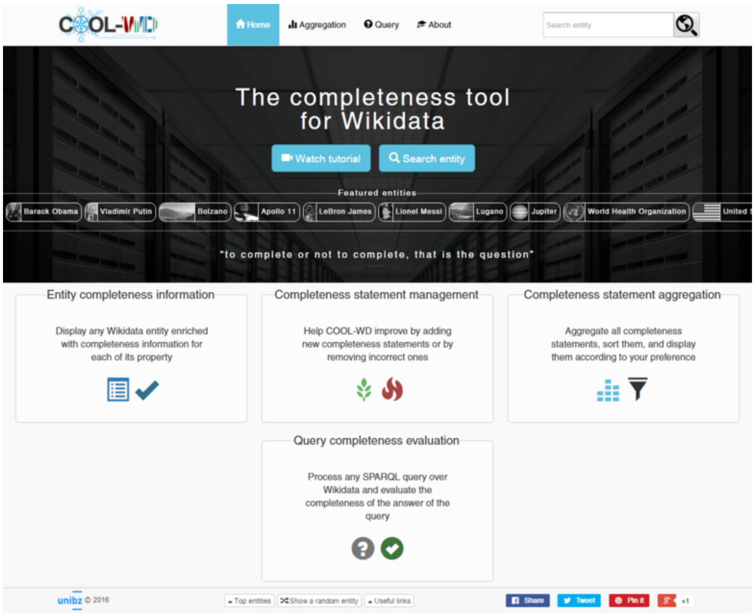
[14]`https://en.wikipedia.org/wiki/Twenty-five_Year_Award`
[15]`https://en.wikipedia.org/wiki/List_of_Italian_DOP_cheeses`

both ends of the data pipeline, production and consumption, is a major reason for the partial closed-world assumption (PCWA) not being adapted on the Semantic Web so far.

In Chapter 7, we develop two demonstrators of systems to manage and consume completeness information, each of which serves different purposes. The first one is CORNER. CORNER demonstrates a completeness statement hub. With CORNER, users may provide completeness statements over multiple RDF data sources and perform data-agnostic completeness reasoning. CORNER supports SPARQL Basic Graph Pattern (BGP) queries and can take RDFS ontologies into account in its analysis. If a query can only be answered completely by a combination of sources, CORNER rewrites the original query into one with SPARQL `SERVICE` calls, which assigns each query part to a suitable source, and executes it over those sources. CORNER can be accessed at `http://corner.inf.unibz.it/`.

The second one is COOL-WD. In contrast to CORNER, COOL-WD demonstrates how one can build a specialized completeness management system over a single KB, in our case, Wikidata. With COOL-WD, end users are provided with web interfaces (available both via the COOL-WD external system and the COOL-WD integrated Wikidata gadget) to create and view completeness information about Wikidata facts. To consume completeness information, COOL-WD users may perform data completion tracking, completeness analytics, or data-aware query completeness assessment with diagnostics. Figure 1.3 shows the homepage of COOL-WD, which can be accessed at `http://cool-wd.inf.unibz.it/`.

Extracting Relation Cardinalities from Text. While CORNER and COOL-WD provide a method to add completeness statements manually, to improve the scalability, an automatic method of generating completeness statements is thus crucial. Meanwhile, over the Web, a wealth of information about relation cardinalities is provided, giving hints on the complete count information of a relation. An example is shown in Figure 1.4 on how cardinality information may produce completeness statements. In this regard, in Chapter 8 we introduce the novel problem of extracting cardinalities from text and analyze specific challenges that set it apart from standard Information Extraction (IE). We present a distant supervision method using conditional random fields (CRF). Our evaluation results in precision between 38% to 84% depending on the difficulty of relations.



The completeness tool for Wikidata

"to complete or not to complete, that is the question"

Entity completeness information

Display any Wikidata entity enriched with completeness information for each of its property

Completeness statement management

Help COOL-WD improve by adding new completeness statements or by removing incorrect ones

Completeness statement aggregation

Aggregate all completeness statements, sort them, and display them according to your preference

Query completeness evaluation

Process any SPARQL query over Wikidata and evaluate the completeness of the answer of the query

Figure 1.3. COOL-WD homepage

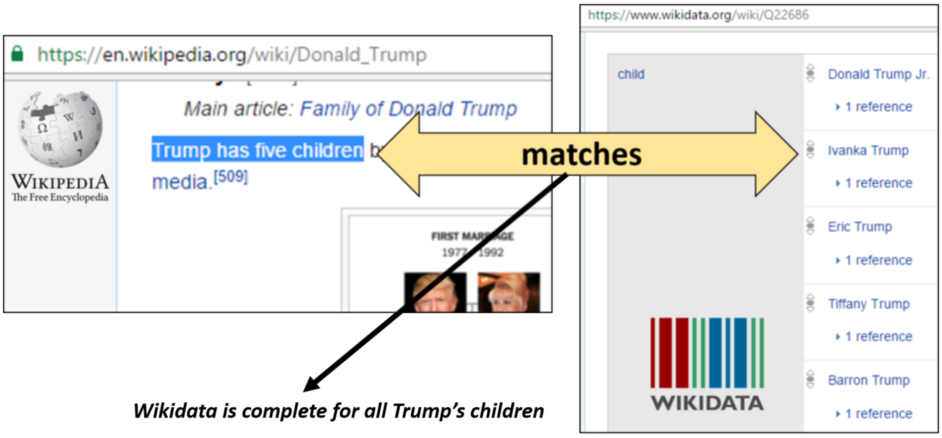

https://en.wikipedia.org/wiki/Donald_Trump
Wikipedia The Free Encyclopedia
Main article: Family of Donald Trump
Trump has five children b
media.[509]
FIRST MARRIAGE
1977 1992
matches
https://www.wikidata.org/wiki/Q22686
child
Donald Trump Jr.
1 reference
Ivanka Trump
1 reference
Eric Trump
1 reference
Tiffany Trump
1 reference
Barron Trump
1 reference
WIKIDATA
Wikidata is complete for all Trump's children

Figure 1.4. By knowing that the children count on Wikidata's Trump page matches the cardinality information from Wikipedia, a completeness statement can be generated

1.3. Contributions

The contributions of this thesis are as follows:

1. we develop a formalization, and a sound and complete algorithm for data-aware completeness reasoning, and explore various practical fragments of completeness statements;
2. we develop optimization techniques for both the data-agnostic and data-aware completeness reasoning, and conduct experimental evaluations based on realistic settings;
3. we formalize the problem of query soundness in the presence of completeness statements, and provide a characterization of the problem via reduction to completeness checking;
4. we introduce time to completeness reasoning;
5. we develop demonstration systems to manage and consume completeness information, that is, CORNER (`http://corner.inf.unibz.it/`) and COOL-WD (`http://cool-wd.inf.unibz.it/`); and
6. we provide a method for extracting relation cardinalities from text on the Web, which can be leveraged to generate completeness statements.

1.4. Thesis Outline

The thesis is structured as follows: Chapter 2 provides some background about RDF and SPARQL, and data-agnostic completeness reasoning for RDF data sources. Chapter 3 discusses data-aware completeness reasoning. In Chapter 4 we propose optimizations of completeness reasoning and report on experimental evaluations of the optimizations. In Chapter 5 we show how our completeness framework can also be leveraged to deal with the problem of query soundness. Chapter 6 extends completeness reasoning with the time information, whereas Chapter 7 describes completeness management demonstrators. Chapter 8 provides an automated approach for extracting relation cardinalities from text on the Web, useful in generating completeness statements. In Chapter 9, we discuss related aspects to our completeness framework. We conclude our work and sketch future directions in Chapter 10.

Chapter 2

Formal Framework

In this chapter, we discuss concepts that are essential for the subsequent content. We remind the reader of RDF and SPARQL in Section 2.1. Section 2.2 formalizes completeness statements, metadata to specify which parts of an RDF data source are complete. We next introduce in Section 2.3 the notion of query completeness. Finally, we define and characterize the completeness entailment problem in the data-agnostic setting in Section 2.4. The results presented in this chapter have been published in [27].

2.1. RDF and SPARQL

We assume 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 3-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.[1] An *RDF graph* G consists of a finite set of triples [59]. For simplicity, we omit namespaces for the abstract representation of RDF graphs.

The standard query language for RDF is SPARQL [46]. The basic building blocks of a SPARQL query are *triple patterns*, which resemble RDF triples, except that in each position also variables are allowed. We focus on the conjunctive fragment of SPARQL, which uses

[1]We do not consider blank nodes in this thesis for the reasons as discussed later in Section 9.2.

sets of triple patterns, called *basic graph patterns* (BGPs).[2] A *mapping* μ is a partial function $\mu\colon V \to I \cup L$. Given a BGP P, μP denotes the BGP obtained by replacing variables in P with terms according to μ. The evaluation of a BGP P over an RDF graph G, denoted as $[\![P]\!]_G$, results in a set of mappings such that for every mapping $\mu \in [\![P]\!]_G$, it holds $\mu P \subseteq G$. For a BGP P, we define the *freeze mapping* $\tilde{id}$ as mapping each variable $?v$ in P to a fresh IRI $\tilde{v}$ (that is, $\tilde{v}$ is a frozen variable). From such a mapping, we construct the *prototypical graph* $\tilde{P} := \tilde{id}\,P$ to represent any possible graph that can satisfy the BGP P. Moreover, we define the mapping with empty domain as the empty mapping $\mu_\emptyset$.

SPARQL queries come as `SELECT`, `ASK`, or `CONSTRUCT` queries. The abstract form of a `SELECT` query is (W, P), where P is a BGP and $W \subseteq \mathit{var}(P)$. A `SELECT` query $Q = (W, P)$ is evaluated over a graph G by projecting the mappings in $[\![P]\!]_G$ to the variables in W, written as $[\![Q]\!]_G = \pi_W([\![P]\!]_G)$. Syntactically, an `ASK` query is a special case of a `SELECT` query where W is empty. A `CONSTRUCT` query has the abstract form (P_1, P_2), where both P_1 and P_2 are BGPs, and $\mathit{var}(P_1) \subseteq \mathit{var}(P_2)$. Evaluating a `CONSTRUCT` query over G yields a graph where P_1 is instantiated with all the mappings in $[\![P_2]\!]_G$. In this thesis, the semantics considered in query evaluation is the bag semantics, which is the default of SPARQL [46]. In bag semantics, duplicates of query answers are kept.

2.2. Completeness Statements

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

Definition 2.1 (Completeness Statement)**.** A *completeness statement* C is defined as $\mathit{Compl}(P_C)$ where P_C is a non-empty BGP.

We use BGPs in order to have a flexibility for representing complex completeness information which requires more than one triple pattern. For example, we express that a source is complete for all pairs of triples that say "*?m is a movie (= Mov) and ?m is directed*

[2]SPARQL with negation will be introduced later in Chapter 5 about soundness reasoning.

(= dir) by Tarantino" using the statement[3]

$$C_{dir} = \mathit{Compl}((?m, a, \mathit{Mov}), (?m, \mathit{dir}, \mathit{tarantino})), \qquad (2.1)$$

whose BGP matches all such pairs. To express that a source is complete for all triples about actors ($=$ *act*) in movies directed by Tarantino, we use

$$C_{act} = \mathit{Compl}((?m, \mathit{act}, ?a), (?m, a, \mathit{Mov}), (?m, \mathit{dir}, \mathit{tarantino})). \qquad (2.2)$$

Now to model the OWA of RDF data sources, we define an extension pair.

Definition 2.2 (Extension Pair)**.** We identify data sources with RDF graphs. Then, adapting a notion introduced by Motro [80], we define an *extension pair* as a pair (G, G') of two graphs, where $G \subseteq G'$. We call G the *available graph* and G' the *ideal graph.*

Here, an available graph is the graph that we currently store, while an ideal graph is a possible extension over the available graph, representing a version of ideal, complete information. Note that by nature, ideal graphs are hypothetical, i.e., data providers or consumers do not need to explicitly deal with G'. In an extension pair, the requirement that G is included in G' formalizes the intuition that the available graph contains no more information than the ideal one (i.e., we assume that available graphs are correct).

Without completeness statements, any graph extending the available graph can be an ideal graph. Having completeness statements restricts the possibilities of ideal graphs: for the parts captured by completeness statements, they must contain no more information than in the available graph. Later on in Section 2.4, we will see that conclusions about query completeness are drawn from these restrictions imposed over ideal graphs. To a statement $C = \mathit{Compl}(P_C)$, we associate the `CONSTRUCT` query $Q_C = (P_C, P_C)$. Note that, given a graph G, the query Q_C returns a graph consisting of those instantiations of the pattern P_C present in G. For example, the query $Q_{C_{act}}$ returns the cast of the Tarantino movies in a graph G. We now define the semantics of completeness statements.

[3]For the sake of readability, we slightly abuse the notation by removing the set brackets of the BGPs of completeness statements.

Definition 2.3 (Satisfaction of Completeness Statements)**.** An extension pair (G, G') satisfies the statement C, written $(G, G') \models C$, if $[\![Q_C]\!]_{G'} \subseteq G$.

Intuitively, an extension pair (G, G') satisfies a completeness statement C, if the subgraph of G' captured by C is also present in G. The above definition naturally extends to the satisfaction of a set $\mathbf{C}$ of completeness statements, that is, $(G, G') \models \mathbf{C}$ iff for all $C \in \mathbf{C}$, it is the case that $[\![Q_C]\!]_{G'} \subseteq G$.

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

$$\begin{aligned} G_{dbp} = \{&(reservoirDogs, dir, tarantino), (pulpFiction, dir, tarantino),\\ &(killBill, dir, tarantino), (desperado, act, tarantino),\\ &(pulpFiction, act, tarantino), (desperado, a, Mov),\\ &(reservoirDogs, a, Mov), (pulpFiction, a, Mov), (killBill, a, Mov)\}. \end{aligned}$$

A possible extension (among others) of the above graph is the graph G'_{dbp}, which additionally contains the information that Tarantino starred in Reservoir Dogs:[4]

$$G'_{dbp} = G_{dbp} \cup \{\, (reservoirDogs, act, tarantino) \,\}.$$

Putting the above two graphs together forms the extension pair (G_{dbp}, G'_{dbp}). In this case, the statement C_{dir} (Eq. 2.1) is satisfied by (G_{dbp}, G'_{dbp}) since all triples from evaluating $Q_{C_{dir}}$ over G'_{dbp} are included in G_{dbp}. In contrast, the statement C_{act} (Eq. 2.2) is not satisfied by (G_{dbp}, G'_{dbp}) because evaluating $Q_{C_{act}}$ over G'_{dbp} returns the triple $(reservoirDogs, act, tarantino)$ that is not in G_{dbp}.

An important tool for characterizing completeness entailment is the transfer operator $T_{\mathbf{C}}$, which captures the complete parts of a graph wrt. a set of completeness statements. Given a set $\mathbf{C}$ of completeness statements and a graph G, the *transfer operator* is defined as

$$T_{\mathbf{C}}(G) = \bigcup_{C \in \mathbf{C}} [\![Q_C]\!]_G. \tag{2.3}$$

The transfer operator takes the union of evaluating over G all the corresponding `CONSTRUCT` queries of the statements in $\mathbf{C}$. In terms

[4]which is actually the case in the real world

of extension pairs, the transfer operator takes the parts of the ideal graph that have to be present in the available graph. In a way, the operator 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 $T_{\mathbf{C}}$ and the definition of the satisfaction of $\mathbf{C}$.

Proposition 2.5 (Properties of $T_{\mathbf{C}}$)**.** *Let* $\mathbf{C}$ *be a set of completeness statements. Then,*

(1) *For every extension pair* (G, G'), $(G, G') \models \mathbf{C}$ *iff* $T_{\mathbf{C}}(G') \subseteq G$.

Consequently, for any graph G *we have that*

(2) *the pair* $(T_{\mathbf{C}}(G), G)$ *is an extension pair satisfying* $\mathbf{C}$*, and*
(3) $T_{\mathbf{C}}(G)$ *is the smallest graph for which this holds.*

Note on completeness statements. In Darari et al. [27] completeness statements are defined slightly differently. There completeness statements may have conditions, which are more general than the unconditional ones. For conditional statements, the instantiations of the conditions are not necessarily included in the graph G. For example, the conditional completeness statement "Complete for all movies under the condition that the movies were directed by Tarantino" differs from the statement C_{dir} above since in the former the graph needs only to contain all such movies $(?m, a, Mov)$ but not the director information wrt. Tarantino $(?m, dir, tarantino)$. We found that this might give some confusion when creating completeness statements. In this thesis, completeness statements generally refer to the unconditional ones (as in Definition 2.1). Nevertheless, conditional completeness statements are still used in Section 5.4 for characterizing a variant of the query soundness problem, and in Section 7.1 about CORNER.

RDF Representation of Completeness Statements. Practically, completeness statements should be compliant with the existing ways of giving metadata about data sources, for instance, by enriching current proposals like `VoID` [6]. Hence, it becomes essential to be able to express completeness statements in RDF. Suppose we want to express that LinkedMDB,[5] an RDF data source about movies, satisfies the

[5] `http://www.linkedmdb.org/`

following completeness statement about all actors in movies directed by Tarantino, as introduced in Eq. (2.2),

$$C_{act} = \mathit{Compl}((?m, \mathit{act}, ?a), (?m, a, \mathit{Mov}), (?m, \mathit{dir}, \mathit{tarantino})).$$

To this end, we need: (*i*) a vocabulary to say that this is a completeness statement about LinkedMDB; (*ii*) a mechanism to state which triple patterns make up the statement's BGP; (*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`, `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 the light of these considerations, we can represent C_{act} in RDF as the following resource `lv:st1`, using Turtle serialization [92].[6]

```
lv:lmdbdataset a void:Dataset ;
  c:hasComplStmt lv:st1 .

lv:st1 a c:CompletenessStatement ;
  c:hasPattern [c:subject [c:varName "m"] ;
    c:predicate s:actor ;
    c:object [c:varName "a"]] ;
  c:hasPattern [c:subject [c:varName "m"] ;
    c:predicate rdf:type ;
    c:object s:Movie] ;
  c:hasPattern [c:subject [c:varName "m"] ;
    c:predicate s:director ;
    c:object dbp:Quentin_Tarantino] .
```

Note that in the Turtle serialization we use unlabeled blank nodes (i.e., anonymous resources), denoted by `[ ... ]`, for reification purposes [84] which do not relate to the semantics of completeness statements.

[6]We provide the prefix declarations in Appendix A.

More generally, consider a statement $Compl(t_1, \dots, t_n)$, where each t_i is a triple pattern. Then, we create a resource to represent the completeness statement, and a resource for each of the t_i that is linked to the statement-resource by the property `hasPattern`. The constituents of each t_i are linked to t_i-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

A usual way to access data is via queries. When querying a data source, we want to know whether the data source provides sufficient information to answer the query, that is, whether the query is *complete* wrt. 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 extension pair a query is complete whenever all answers we retrieve over the ideal state are also retrieved over the available state. We now define query completeness wrt. extension pairs.

Definition 2.6 (Query Completeness)**.** Let Q be a `SELECT` query. To express that Q *is complete*, we write $Compl(Q)$. An extension pair (G, G') satisfies $Compl(Q)$, if the result of Q evaluated over G' also appears in Q over G, that is, $[\![Q]\!]_{G'} \subseteq [\![Q]\!]_G$.[7] In this case we write $(G, G') \models Compl(Q)$.

The above definition can be naturally adapted for the completeness of a BGP P, written $Compl(P)$, that is used in later chapters: An extension pair (G, G') satisfies $Compl(P)$, written $(G, G') \models Compl(P)$, if $[\![P]\!]_{G'} \subseteq [\![P]\!]_G$.

Example 2.7. Consider the extension pair (G_{dbp}, G'_{dbp}) and the two queries Q_{dir}, asking for all movies directed by Tarantino, and $Q_{dir+act}$, asking for all movies both directed by and starring Tarantino,

$$\begin{aligned} Q_{dir} &= (\{\, ?m \,\}, \{\, (?m, a, Mov), (?m, dir, tarantino) \,\}), \text{ and} \\ Q_{dir+act} &= (\{\, ?m \,\}, \{\, (?m, a, Mov), (?m, dir, tarantino), \\ &\qquad (?m, act, tarantino) \,\}). \end{aligned}$$

[7]For monotonic queries, the other direction, that is, $[\![Q]\!]_{G'} \supseteq [\![Q]\!]_G$, comes for free. Hence, we sometimes use the '=' condition when queries are monotonic.

Then, it holds that Q_{dir} is complete over (G_{dbp}, G'_{dbp}) since it is the case that $[\![Q_{dir}]\!]_{G_{dbp}} = \{\,\{\,?m \mapsto \mathit{reservoirDogs}\,\},\ \{\,?m \mapsto \mathit{pulpFiction}\,\}, \{\,?m \mapsto \mathit{killBill}\,\}\,\} = [\![Q_{dir}]\!]_{G'_{dbp}}$. On the other hand, $Q_{dir+act}$ is *not* complete over (G_{dbp}, G'_{dbp}) since $[\![Q_{dir+act}]\!]_{G_{dbp}}$ does not contain the result mapping $\{\,?m \mapsto \mathit{reservoirDogs}\,\}$, which occurs in $[\![Q_{dir+act}]\!]_{G'_{dbp}}$.

2.4. Data-agnostic Completeness Entailment

From the notions above, a question naturally arises as to when some meta-information about data completeness can provide a guarantee for query completeness. In other words, the available state contains all data, as guaranteed by the completeness statements, that is required for computing the query answer, so one can trust the results of the query. While previously we have looked at examples with concrete extension pairs, in the following we formalize the completeness entailment problem in the data-agnostic setting, that is, when the available graph to which completeness statements are given is also abstracted (recall that ideal graphs are always abstracted). This way, we 'quantify' over all extension pairs such that if an extension pair satisfies the completeness statements, then it must also satisfy the query completeness.

Definition 2.8 (Data-agnostic Completeness Entailment)**.** Let $\mathbf{C}$ be a set of completeness statements and Q be a `SELECT` query. We say that $\mathbf{C}$ *entails the completeness of* Q, written $\mathbf{C} \models \mathit{Compl}(Q)$, if any extension pair satisfying $\mathbf{C}$ also satisfies $\mathit{Compl}(Q)$.

Example 2.9. Consider C_{dir} from Eq. (2.1). Whenever an extension pair (G, G') satisfies C_{dir}, then G contains all triples about movies directed by Tarantino, which is exactly the information needed to answer Q_{dir} from Example 2.7. Thus, $\{\,C_{dir}\,\} \models \mathit{Compl}(Q_{dir})$. However, C_{dir} is not enough to completely answer $Q_{dir+act}$, thus $\{\,C_{dir}\,\} \not\models \mathit{Compl}(Q_{dir+act})$.

We want to provide a characterization of the entailment. To check whether the completeness of a query $Q = (W, P)$ 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 in the evaluation result, we have $\tilde{P}$ back. Intuitively, this means that over any possible graph instantiation for answering the query, the completeness statements guarantee that we have back the graph instantiation in our available data source. The following theorem characterizes the completeness of SPARQL queries.

Theorem 2.10 (Completeness of `SELECT` Queries [27]). *Let $\boldsymbol{C}$ be a set of completeness statements and $Q = (W, P)$ be a `SELECT` query. Then,*

$$\boldsymbol{C} \models \mathit{Compl}(Q) \qquad \mathit{iff} \qquad \tilde{P} = T_{\boldsymbol{C}}(\tilde{P}).$$

The following complexity result [27] follows as the completeness check is basically evaluating a linear number of `CONSTRUCT` queries over the (frozen) conjunctive body of the query.

Corollary 2.11. *Deciding whether* $\mathbf{C} \models \mathit{Compl}(Q)$*, given a set* $\mathbf{C}$ *of completeness statements and a `SELECT` query* $Q = (W, P)$*, 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 [19].

Chapter 3

Data-aware Completeness Reasoning

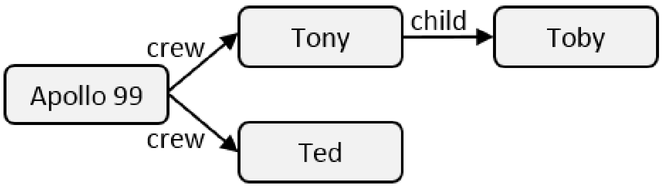
In the previous chapter, we have formalized completeness information, and characterized its use for checking query completeness in the data-agnostic setting. Data-agnostic completeness checking takes a set of completeness statements and a query as input parameters, and says whether the query can be guaranteed to be complete. In such checking, the available graph for which completeness statements are applied is not taken into account. As a consequence, data-specific inferences cannot be drawn. Yet, since completeness statements are generally created within the context of an available graph, query completeness may also depend on the graph.

In this chapter, we tackle the problem of completeness checking in the data-aware setting, that is, given a set of completeness statements, a query, *and* an RDF graph, we check whether the completeness of the query can be guaranteed. This chapter is divided into the following sections. Section 3.1 gives a motivating scenario of data-aware completeness reasoning. Section 3.2 formalizes the problem of data-aware completeness entailment and provides a characterization of the problem. Section 3.3 introduces SP-statements, a fragment of completeness statements that is suitable for entity-centric, crowdsourced RDF data sources, while Section 3.4 introduces no-value statements, a fragment of completeness statements that concerns the non-existence of information in RDF. Related work is given in Section 3.5. We summarize this chapter in Section 3.6.

The results of this chapter have been published in [33] for the parts of data-aware completeness entailment as well as SP-statements, and in [31] for no-value statements.

3.1. Motivating Scenario

Let us consider a motivating scenario for the main problem of this chapter, that is, the checking of query completeness based on RDF data with completeness information. Consider an RDF graph G about the crew of Apollo 99 (or for short, A99), a fictional space mission, and the children of the crew, as displayed below.

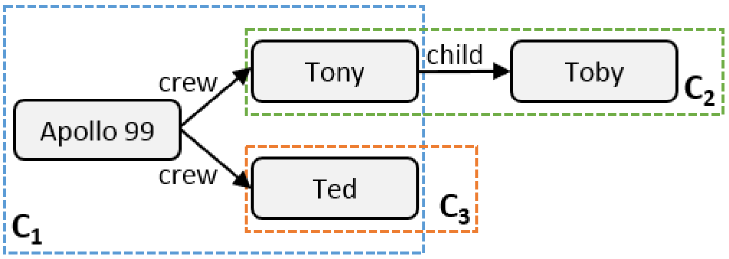


Consider now the query Q_0 asking for the crew of A99 and their children:

$$Q_0 = (W_0, P_0) = (\{\,?crew, ?child\,\}, \{\,(a99, crew, ?crew),\\ (?crew, child, ?child)\}).$$

Evaluating Q_0 over the graph gives only one mapping result, where the crew is mapped to Tony and the child is mapped to Toby. Up until now, nothing can be said about the completeness of the query since: (*i*) there can be another crew member of A99 with a child; (*ii*) Tony may have another child; or (*iii*) Ted may have a child.

Let us consider the same graph as before, now enriched with completeness information, as shown below.



The above figure illustrates three completeness statements:

- $C_1 = Compl((a99, crew, ?c))$, which states that the graph contains all crew members of A99;

- $C_2 = \mathit{Compl}((\mathit{tony}, \mathit{child}, ?c))$, which states the graph contains all Tony's children; and
- $C_3 = \mathit{Compl}((\mathit{ted}, \mathit{child}, ?c))$, which states the graph contains all Ted's children (i.e., Ted has no children).

With the addition of this completeness information, let us see whether we can answer our query completely.

First, from the completeness statement C_1 about all A99 crew, we can infer that the part $(\mathit{a99}, \mathit{crew}, ?\mathit{crew})$ of Q_0 is complete. By evaluating that part over G, we know that all the A99 crew members are Tony and Ted. In terms of extension pairs, that means that no extension $G' \supseteq G$ satisfying C_1 has other A99 crew members than Tony and Ted. In summary, this allows us to instantiate the query Q_0 into the following two queries that are intuitively equivalent with Q_0 itself:

- $Q_1 = (W_1, P_1) = (\{?\mathit{child}\}, \{(\mathit{a99}, \mathit{crew}, \mathit{tony}), (\mathit{tony}, \mathit{child}, ?\mathit{child})\})$
- $Q_2 = (W_2, P_2) = (\{?\mathit{child}\}, \{(\mathit{a99}, \mathit{crew}, \mathit{ted}), (\mathit{ted}, \mathit{child}, ?\mathit{child})\})$

where we record that the variable $?\mathit{crew}$ has been assigned to Tony and Ted, respectively.

Our task is now transformed to checking whether Q_1 and Q_2 can be answered completely. As for Q_2, we know that from the statement C_3, we are complete for the part $(\mathit{ted}, \mathit{child}, ?\mathit{child})$. This again allows us to instantiate the query Q_2 wrt. the graph G. However, now we come to the situation where there is no applicable part in G: instantiating the part $(\mathit{ted}, \mathit{child}, ?\mathit{child})$ gives nothing (i.e., Ted has no children). In other words, for any possible extension G' of G, as guaranteed by C_3, the extension G' is also empty for the part $(\mathit{ted}, \mathit{child}, ?\mathit{child})$. Thus, there is no way that Q_2 will return an answer, so Q_2 can be safely removed. In a way, we can also see that we are complete for Q_2.

Now, only the query Q_1 is left. Again, from the statement C_2, we know that we are complete for the part $(\mathit{tony}, \mathit{child}, ?\mathit{child})$ of Q_1. This allows us to instantiate the query Q_1 into the query Q_3 that is intuitively equivalent with Q_1 itself:

$$Q_3 = (W_3, P_3) = (\{\,\}, \{\,(\mathit{a99}, \mathit{crew}, \mathit{tony}), (\mathit{tony}, \mathit{child}, \mathit{toby})\,\}),$$

where we record that the variable $?\mathit{crew}$ has been assigned to Tony and ?*child* to Toby. However, our graph is complete for Q_3 as it contains the whole ground body of Q_3. In this case, no extension G' of G can contain more information about Q_3. Now, tracing back our reasoning steps, we know that our Q_3 is in fact intuitively equivalent to our original query Q_0. Since we are complete for Q_3, we are also complete for Q_0, wrt. our graph and completeness statements. In other words, our statements and graph can guarantee the completeness of the query Q_0. Concretely, this means that Toby is the only child of Tony, the only crew member of A99 with a child.

To generalize our example, we have reasoned about the completeness of a query given a set of completeness statements and a graph. The reasoning is basically done as follows: First we find parts of the query that can be guaranteed to be complete by the completeness statements. Then, we produce equivalent query instantiations by evaluating those complete query parts over the graph and applying the obtained mappings to the query itself. Next, for all the query instantiations, we repeat the above steps until no further complete parts can be found. The original query is complete iff all the BGPs of the generated queries are contained in the data graph.

Note that using the data-agnostic approach as in Section 2.4, it is not possible to derive the same conclusion. Without looking at the actual graph, we cannot conclude that Ted and Tony are all the crew members of Apollo 99, that is, it can even be that all the crew members are completely different people like Bob, John, and Romeo. Consequently, just having the children of Tony and Ted complete does not help reason about Apollo 99.

In the next section, we discuss how the intuitive, data-specific reasoning from above can be formalized.

3.2. Checking Data-aware Completeness Entailment

In contrast to data-agnostic completeness entailment, in data-aware completeness entailment, the specifics of the graph matter, as formalized below.

Definition 3.1 (Data-aware Completeness Entailment)**.** Given a set $\mathbf{C}$ of completeness statements, a graph G, and a query Q, we define that $\mathbf{C}$ *and* G *entail the completeness of* Q, written as $\mathbf{C}, G \models \mathit{Compl}(Q)$, if for all extension pairs $(G, G') \models \mathbf{C}$, it is the case that $(G, G') \models \mathit{Compl}(Q)$.

As we assume bag semantics for query evaluation, we can therefore focus on the BGPs used in the body of queries for completeness entailment. The following proposition provides an initial characterization of completeness entailment as a reference on how to develop formal notions and an algorithm for completeness checking. Basically, for a set of completeness statements, a graph, and a BGP, the completeness entailment holds, iff extending the graph with a possible BGP instantiation (wrt. some mapping) such that the extension satisfies the statements, will always result in the inclusion of the BGP instantiation in the graph itself.

Proposition 3.2. *Let* $\mathbf{C}$ *be a set of completeness statements,* G *be a graph, and* P *be a BGP. Then, it holds that:* $\mathbf{C}, G \models \mathit{Compl}(P)$ *iff for every mapping* μ *such that* $\mathit{dom}(\mu) = \mathit{var}(P)$ *and* $(G, G \cup \mu P) \models \mathbf{C}$*, it is the case that* $\mu P \subseteq G$*.*

Proof. ($\Rightarrow$) We prove by contrapositive. Suppose there is a mapping μ where $\mathit{dom}(\mu) = \mathit{var}(P)$ and $(G, G \cup \mu P) \models \mathbf{C}$, but $\mu P \nsubseteq G$. We want to show $\mathbf{C}, G \not\models \mathit{Compl}(P)$. For this, we need a counterexample extension pair (G, G') such that $(G, G') \models \mathbf{C}$ but $(G, G') \not\models \mathit{Compl}(P)$.

Take the extension pair $(G, G \cup \mu P)$. By assumption, we have that $(G, G \cup \mu P) \models \mathbf{C}$. Now let us see whether $(G, G \cup \mu P) \models \mathit{Compl}(P)$ or not. Again, by assumption we have that $\mu P \nsubseteq G$. This means that $\mu \notin [\![P]\!]_G$ despite the obvious case that $\mu \in [\![P]\!]_{G \cup \mu P}$. This implies that $(G, G \cup \mu P) \not\models \mathit{Compl}(P)$. Therefore, $\mathbf{C}, G \not\models \mathit{Compl}(P)$ as witnessed by the counterexample extension pair $(G, G \cup \mu P)$.
($\Leftarrow$) Assume that for all mappings μ such that $\mathit{dom}(\mu) = \mathit{var}(P)$ and $(G, G \cup \mu P) \models \mathbf{C}$, it is the case $\mu P \subseteq G$. We want to show that $\mathbf{C}, G \models \mathit{Compl}(P)$. Take an extension pair (G, G') such that $(G, G') \models \mathbf{C}$. We need to prove that $(G, G') \models \mathit{Compl}(P)$. In other words, it has to be shown that $[\![P]\!]_{G'} \subseteq [\![P]\!]_G$.

Now take a mapping $\mu \in [\![P]\!]_{G'}$. By the semantics of BGP evaluation, this implies $\mu P \subseteq G'$. We want to show $\mu \in [\![P]\!]_G$. Again, by the semantics of BGP evaluation it is sufficient to show $\mu P \subseteq G$. By the assumption that $(G, G') \models \mathbf{C}$ and the semantics of the $T_{\mathbf{C}}$ operator, we have that $T_{\mathbf{C}}(G') \subseteq G$. From this and $\mu P \subseteq G'$ (and also $G \subseteq G'$ by the definition of an extension pair), it holds that $T_{\mathbf{C}}(G \cup \mu P) \subseteq T_{\mathbf{C}}(G') \subseteq G$. Therefore, it is the case that $(G, G \cup \mu P) \models \mathbf{C}$. By assumption, it is the case $\mu P \subseteq G$. Since μ was arbitrary, we can therefore conclude that $[\![P]\!]_{G'} \subseteq [\![P]\!]_G$. $\square$

In other words, the completeness entailment does not hold, iff we can find a possible BGP instantiation (wrt. some mapping) such that the extension satisfies the statements, but the BGP instantiation is not contained in the graph. The idea here is that, as demonstrated in our motivating example, by using completeness statements we always try to find complete parts of the BGP and instantiate them over the graph, until either all the instantiations are included in the graph (= the success case), or there is one instantiation that is not included there (= the failure case). In the following subsections, we provide formal notions and an algorithm for checking data-aware completeness entailment.

3.2.1. Formal Notions

We now introduce formal notions to be used later in our algorithm for checking data-aware completeness entailment.

First, we need a notion for a BGP with a stored mapping from variable instantiations. This allows us to represent BGP instantiations wrt. our completeness entailment procedure. Let P be a BGP and μ be a mapping such that $dom(\mu) \cap var(P) = \emptyset$. We define the pair (P, μ) as a *partially mapped BGP*, which is a BGP with a stored mapping. Over a graph G, the evaluation of (P, μ) is defined as $[\![(P, \mu)]\!]_G = \{\, \mu \cup \nu \mid \nu \in [\![P]\!]_G \,\}$. It is easy to see that $P \equiv (P, \emptyset)$. Furthermore, we define the evaluation of a set of partially mapped BGPs over a graph G as the union of evaluating each of them over G.

Example 3.3. Consider our motivating scenario. Over the BGP P_0 of the query Q_0, instantiating the variable $?crew$ to $tony$ results in the BGP P_1 of the query Q_1. Pairing P_1 with this instantiation gives the partially mapped BGP $(P_1, \{\, ?crew \mapsto tony \,\})$. Moreover, it is the case that $[\![(P_1, \{\, ?crew \mapsto tony \,\})]\!]_G = \{\, \{\, ?crew \mapsto tony, ?child \mapsto toby \,\} \,\}$.

Next, we want to formalize the equivalence between partially mapped BGPs wrt. a set $\mathbf{C}$ of completeness statements and a graph G. We need this notion to ensure the equivalence of the BGP instantiations that resulted from the evaluation of complete BGP parts.

Definition 3.4 (Equivalence under $\mathbf{C}$ and G)**.** Let (P, μ) and (P', ν) be partially mapped BGPs, $\mathbf{C}$ be a set of completeness statements, and G be a graph. We define that (P, μ) is *equivalent* to (P', ν) wrt.

$\mathbf{C}$ and G, written $(P, \mu) \equiv_{\mathbf{C},G} (P', \nu)$, if for all $(G, G') \models \mathbf{C}$, it holds that $[\![(P, \mu)]\!]_{G'} = [\![(P', \nu)]\!]_{G'}$.

The above definition naturally extends to sets of partially mapped BGPs.

Example 3.5. Consider all the queries in our motivating scenario. It is the case that:

$$\{\, (P_0, \emptyset) \,\} \equiv_{\mathbf{C},G} \{\, (P_1, \{\, ?crew \mapsto tony \}), (P_2, \{\, ?crew \mapsto ted \}) \,\} \equiv_{\mathbf{C},G} \{\, (P_3, \{\, ?crew \mapsto tony, ?child \mapsto toby \}) \,\}.$$

Next, we would like to figure out which parts of a BGP contain variables that can be instantiated completely. The idea is that, we 'match' completeness statements to the BGP and the graph, and return the matched parts of the BGP. Note that in the matching we consider also the graph since it might be the case that for a single completeness statement, some parts of it have to be matched to the BGP, while the other parts to the graph. For this reason, we define

$$cruc_{\mathbf{C},G}(P) = P \cap \tilde{id}^{-1}(T_{\mathbf{C}}(\tilde{P} \cup G)) \tag{3.1}$$

as the *crucial part of P wrt. $\mathbf{C}$ and G*. It is the case that we are complete for the crucial part, that is, $\mathbf{C}, G \models Compl(cruc_{\mathbf{C},G}(P))$. Later on, we will see that the crucial part is used to guide the instantiation process during the completeness entailment check.

Example 3.6. Consider the query $Q_0 = (W_0, P_0)$ in our motivating scenario. We have that

$$cruc_{\mathbf{C},G}(P_0) = P_0 \cap \tilde{id}^{-1}(T_{\mathbf{C}}(\tilde{P}_0 \cup G)) = \{\, (a99, crew, ?crew) \,\}$$

with $\tilde{id} = \{\, ?crew \mapsto \widetilde{crew}, ?child \mapsto \widetilde{child} \,\}$. Consequently, we can have a complete instantiation of the crew of A99.

The operator below implements the instantiations of a partially mapped BGP wrt. its crucial part.

Definition 3.7 (Equivalent Partial Grounding)**.** Let $\mathbf{C}$ be a set of completeness statements, G be a graph, and (P, ν) be a partially mapped BGP. We define the operator *equivalent partial grounding:*

$$epg((P, \nu), \mathbf{C}, G) = \{\, (\mu P, \nu \cup \mu) \mid \mu \in [\![cruc_{\mathbf{C},G}(P)]\!]_G \,\}.$$

The following shows that such instantiations produce a set of partially mapped BGPs equivalent to the original partially mapped BGP, hence the name equivalent partial grounding. Basically, it holds since the instantiation is done over the crucial part, which is complete wrt. $\mathbf{C}$ and G.

Proposition 3.8 (Equivalent Partial Grounding). *Let $\mathbf{C}$ be a set of completeness statements, G be a graph, and (P, ν) be a partially mapped BGP. Then,*

$$\{\, (P, \nu) \,\} \equiv_{\mathbf{C},G} \mathit{epg}((P, \nu), \mathbf{C}, G).$$

Proof. Take any G' such that $(G, G') \models \mathbf{C}$. We want to show that $[\![(P, \nu)]\!]_{G'} = \bigcup_{(\mu P, \nu \cup \mu) \in \mathit{epg}((P,\nu),\mathbf{C},G)} [\![(\mu P, \nu \cup \mu)]\!]_{G'}$. Since it is the case $\mathit{dom}(\nu) \cap \mathit{var}(P) = \emptyset$ by the construction of a partially mapped BGP, it is sufficient to show that $[\![(P, \emptyset)]\!]_{G'} = \bigcup_{(\mu P, \mu) \in \mathit{epg}((P,\emptyset),\mathbf{C},G)} [\![(\mu P, \mu)]\!]_{G'}$. By the construction of the *epg* operator, it is enough to show that $[\![(P, \emptyset)]\!]_{G'} = \bigcup_{\mu \in [\![\mathit{cruc}_{\mathbf{C},G}(P)]\!]_G} [\![(\mu P, \mu)]\!]_{G'}$.

Recall that the crucial part of P is complete wrt. $\mathbf{C}$ and G, that is, $\mathbf{C}, G \models \mathit{Compl}(\mathit{cruc}_{\mathbf{C},G}(P))$. This implies that $[\![\mathit{cruc}_{\mathbf{C},G}(P)]\!]_G = [\![\mathit{cruc}_{\mathbf{C},G}(P)]\!]_{G'}$. Therefore, it is the case $\bigcup_{\mu \in [\![\mathit{cruc}_{\mathbf{C},G}(P)]\!]_G} [\![(\mu P, \mu)]\!]_{G'} = \bigcup_{\mu \in [\![\mathit{cruc}_{\mathbf{C},G}(P)]\!]_{G'}} [\![(\mu P, \mu)]\!]_{G'}$. By construction, it is always the case $\mathit{cruc}_{\mathbf{C},G}(P) \subseteq P$. Given this fact and the semantics of evaluating a partially mapped BGP, it holds that $\bigcup_{\mu \in [\![\mathit{cruc}_{\mathbf{C},G}(P)]\!]_{G'}} [\![(\mu P, \mu)]\!]_{G'} = [\![(P, \emptyset)]\!]_{G'}$. Thus, we can conclude that $\bigcup_{\mu \in [\![\mathit{cruc}_{\mathbf{C},G}(P)]\!]_G} [\![(\mu P, \mu)]\!]_{G'} = \bigcup_{\mu \in [\![\mathit{cruc}_{\mathbf{C},G}(P)]\!]_{G'}} [\![(\mu P, \mu)]\!]_{G'} = [\![(P, \emptyset)]\!]_{G'}$.

□

Example 3.9. Consider our motivating scenario. We have that:

- $\mathit{epg}((P_2, \{\, ?\mathit{crew} \mapsto \mathit{ted} \,\}), \mathbf{C}, G) = \emptyset$
- $\mathit{epg}((P_3, \{\, ?\mathit{crew} \mapsto \mathit{tony}, ?\mathit{child} \mapsto \mathit{toby} \,\}), \mathbf{C}, G)$ $= \{(P_3, \{?\mathit{crew} \mapsto \mathit{tony}, ?\mathit{child} \mapsto \mathit{toby}\})\}$
- $\mathit{epg}((P_0, \emptyset), \mathbf{C}, G) = \{\, (P_1, \{\, ?\mathit{crew} \mapsto \mathit{tony} \,\}), (P_2, \{\, ?\mathit{crew} \mapsto \mathit{ted} \,\}) \,\}$

Generalizing from the example above, there are three cases of the operator $\mathit{epg}((P, \nu), \mathbf{C}, G)$:

- If $[\![\mathit{cruc}_{\mathbf{C},G}(P)]\!]_G = \emptyset$, it returns the empty set.
- If $[\![\mathit{cruc}_{\mathbf{C},G}(P)]\!]_G = \{\, \mu_\emptyset \,\}$, it returns $\{(P, \nu)\}$.

- Otherwise, it returns a non-empty set of partially mapped BGPs where some variables in P are instantiated.

From these three cases and the finite number of triple patterns with variables of a BGP, it holds that the repeated applications of the *epg* operator, with the first and second cases above as the base cases, are terminating. Note that the difference between these two base cases is in the effect of their corresponding *epg* operations, as illustrated in Example 3.9: for the first case, the *epg* operation returns the empty set, whereas for the second case, it returns back the input partially mapped BGP. Also, intuitively the first case corresponds to the non-existence of the query answer in any possible extension of the graph that satisfies the set of completeness statements (e.g., the Ted's children case).

As for the second case, we need a different treatment. We first define that a partially mapped BGP (P, ν) is *saturated* wrt. $\mathbf{C}$ and G, if $epg((P, \nu), \mathbf{C}, G) = \{ (P, \nu) \}$, that is, if the second case above applies. Note that the notion of saturation is independent from the mapping in a partially mapped BGP: given a mapping ν, a partially mapped BGP (P, ν) is saturated wrt. $\mathbf{C}$ and G iff (P, ν') is saturated wrt. $\mathbf{C}$ and G for any mapping ν'. Thus, wrt. $\mathbf{C}$ and G we say that a BGP P is saturated if $(P, \emptyset)$ is saturated.

Saturated BGPs hold the key as to whether our completeness entailment succeeds or not: the completeness checking of saturated BGPs is simply by checking whether they are contained in the graph G.

Lemma 3.10 (Completeness Entailment of Saturated BGPs)**.** *Let P be a BGP, $\mathbf{C}$ be a set of completeness statements, and G be a graph. Suppose P is saturated wrt. $\mathbf{C}$ and G. Then, it is the case that: $\mathbf{C}, G \models Compl(P)$ iff $\tilde{P} \subseteq G$.*

Proof. ($\Rightarrow$) We prove by contrapositive. Suppose $\tilde{P} \nsubseteq G$. We want to give a counterexample for $\mathbf{C}, G \models Compl(P)$. Let us take the extension pair $(G, G \cup \tilde{P})$. Note that since $\tilde{P} \nsubseteq G$, it is the case that $[\![P]\!]_{G \cup \tilde{P}} \nsubseteq [\![P]\!]_G$, implying $(G, G \cup \tilde{P}) \not\models Compl(P)$.

It is left to show $(G, G \cup \tilde{P}) \models \mathbf{C}$. We would like to prove the following: If P is saturated wrt. $\mathbf{C}$ and G, then $(G, G \cup \tilde{P}) \models \mathbf{C}$. By definition, wrt. $\mathbf{C}$ and G a BGP P is saturated iff $(P, \emptyset)$ is saturated. From our assumption that P is saturated, we therefore know that $(P, \emptyset)$ is also saturated. By the definition of saturation, this means that $epg((P, \emptyset), \mathbf{C}, G) = \{ (P, \emptyset) \}$. This implies that $[\![cruc_{\mathbf{C},G}(P)]\!]_G =$

$\{\mu_\emptyset\}$. Consequently, $\mu_\emptyset(cruc_{\mathbf{C},G}(P)) = cruc_{\mathbf{C},G}(P) \subseteq G$. Here we know that $cruc_{\mathbf{C},G}(P)$ is ground.

Now we want show that $T_{\mathbf{C}}(\tilde{P} \cup G) \subseteq G$ for the following reason: by the definition of $T_{\mathbf{C}}$ and the satisfaction of an extension pair wrt. $\mathbf{C}$, it is the case that $T_{\mathbf{C}}(\tilde{P} \cup G) \subseteq G$ implies $(G, \tilde{P} \cup G) \models \mathbf{C}$.

By construction, the $T_{\mathbf{C}}$ operator always returns a subset of the input. There are therefore two components of the results of $T_{\mathbf{C}}(\tilde{P} \cup G)$ we have to check if they are included in G. The first is those included in G, that is, $G \cap T_{\mathbf{C}}(\tilde{P} \cup G)$. Clearly, $G \cap T_{\mathbf{C}}(\tilde{P} \cup G) \subseteq G$.

The second one is those included in $\tilde{P}$, that is, $\tilde{P} \cap T_{\mathbf{C}}(\tilde{P} \cup G)$. We want to show that $\tilde{P} \cap T_{\mathbf{C}}(\tilde{P} \cup G) \subseteq G$. Recall that $cruc_{\mathbf{C},G}(P) \subseteq G$. By definition, $cruc_{\mathbf{C},G}(P) = P \cap \tilde{id}^{-1}(T_{\mathbf{C}}(\tilde{P} \cup G))$. Since $cruc_{\mathbf{C},G}(P)$ is ground, we have that $cruc_{\mathbf{C},G}(P) = \tilde{P} \cap \tilde{id}^{-1}(T_{\mathbf{C}}(\tilde{P} \cup G))$, and the melting operator $\tilde{id}^{-1}$ does not have any effect, that is, $\tilde{P} \cap \tilde{id}^{-1}(T_{\mathbf{C}}(\tilde{P} \cup G)) = \tilde{P} \cap (T_{\mathbf{C}}(\tilde{P} \cup G))$. Consequently, we have $cruc_{\mathbf{C},G}(P) = \tilde{P} \cap (T_{\mathbf{C}}(\tilde{P} \cup G)) \subseteq G$.

Since both components are in G, we have that $T_{\mathbf{C}}(\tilde{P} \cup G) \subseteq G$, and therefore $(G, \tilde{P} \cup G) \models \mathbf{C}$.
($\Leftarrow$) Assume $\tilde{P} \subseteq G$. It is trivial to see that P is ground (i.e., has no variables), and $P \subseteq G$. Therefore, it is the case that for all extension pairs (G, G'), the inclusion $[\![P]\!]_{G'} \subseteq [\![P]\!]_G$ holds, implying $(G, G') \models Compl(P)$. By definition, $\mathbf{C}, G \models Compl(P)$ holds if for all $(G, G') \models \mathbf{C}$, we have $(G, G') \models Compl(P)$. Hence, $\mathbf{C}, G \models Compl(P)$ holds since $(G, G') \models Compl(P)$ even for all possible extension pairs (G, G'). □

By consolidating all the above notions, we are ready to provide an algorithm to check data-aware completeness entailment. The next subsection presents the algorithm.

3.2.2. Algorithm

From the above notions, we have defined the *cruc* operator, useful to find parts of a BGP that can be instantiated completely. The instantiation process wrt. the crucial part is facilitated by the *epg* operator. We have also learned that repeating the application of the *epg* operator results in saturated BGPs for which we have to check whether they are contained in the graph or not, in order to know whether our original BGP is complete. Let us now introduce an algorithm to compute, given a set of completeness statements $\mathbf{C}$, a graph G, and a BGP P, all mappings that have two properties: each BGP instantiation of the mappings constitutes a saturated BGP wrt. $\mathbf{C}$ and G; and the original BGP is equivalent wrt. $\mathbf{C}$ and G with the BGP instantiations produced from all the resulting mappings of the algorithm.

ALGORITHM 1: $\mathit{sat}(P_{orig}, \mathbf{C}, G)$

```
Input: A BGP $P_{orig}$, a set $\mathbf{C}$ of completeness statements, a graph $G$
Output: A set $\Omega$ of mappings
$\mathbf{P}_{working} \leftarrow \{ (P_{orig}, \emptyset) \}$
$\Omega \leftarrow \emptyset$
while $\mathbf{P}_{working} \neq \emptyset$ do
    $(P, \nu) \leftarrow$ takeOne($\mathbf{P}_{working}$)
    $\mathbf{P}_{equiv} \leftarrow \mathit{epg}((P, \nu), \mathbf{C}, G)$
    if $\mathbf{P}_{equiv} = \{ (P, \nu) \}$ then
        $\Omega \leftarrow \Omega \cup \nu$
    else
        $\mathbf{P}_{working} \leftarrow \mathbf{P}_{working} \cup \mathbf{P}_{equiv}$
    end
end
return $\Omega$
```

Consider a BGP P_{orig}, a set $\mathbf{C}$ of completeness statements, and a graph G. The algorithm works as follows: First, we transform our original BGP P_{orig} into its equivalent partially mapped BGP $(P_{orig}, \emptyset)$ and put it in $\mathbf{P}_{working}$. Then, in each iteration of the while loop, we take and remove a partially mapped BGP (P, ν) from $\mathbf{P}_{working}$ via the method `takeOne`. Afterwards, we compute $\mathit{epg}((P, \nu), \mathbf{C}, G)$. As discussed above there might be three result cases here: (i) If $\mathit{epg}((P, \nu), \mathbf{C}, G) = \emptyset$, then simply we remove (P, ν) and will not consider it anymore in the later iteration; (ii) If $\mathit{epg}((P, \nu), \mathbf{C}, G) = \{ (P, \nu) \}$, that is, (P, ν) is saturated, then we collect the mapping ν to the set Ω; and (iii) otherwise, we add to $\mathbf{P}_{working}$ a set of partially mapped BGPs instantiated from (P, ν). We keep iterating until $\mathbf{P}_{working} = \emptyset$, and finally return the set Ω.

The following proposition follows from the construction of the above algorithm and Proposition 3.8.

Proposition 3.11. *Given a BGP P, a set $\mathbf{C}$ of completeness state-*

ments, and a graph G, the following properties hold:

- *For all $\mu \in sat(P, \mathbf{C}, G)$, it is the case that μP is saturated wrt. $\mathbf{C}$ and G.*
- *It holds that $\{(P, \emptyset)\} \equiv_{\mathbf{C},G} \{\, (\mu P, \mu) \mid \mu \in sat(P, \mathbf{C}, G) \,\}$.*

From the above proposition, we can derive the following theorem, which shows the soundness and completeness of the algorithm to check completeness entailment.

Theorem 3.12 (Completeness Entailment Check)**.** *Let P be a BGP, $\mathbf{C}$ be a set of completeness statements, and G be a graph. It holds that*

$$\mathbf{C}, G \models Compl(P) \quad \textit{iff} \quad \textit{for all } \mu \in sat(P, \mathbf{C}, G)\ .\ \widetilde{\mu P} \subseteq G.$$

Proof. ($\Rightarrow$) We prove by contrapositive. Assume there exists a mapping $\mu \in sat(P, \mathbf{C}, G)$ such that $\widetilde{\mu P} \nsubseteq G$. From Proposition 3.11, we have that μP is saturated wrt. $\mathbf{C}$ and G. From Lemma 3.10, it is the case $\mathbf{C}, G \not\models Compl(\mu P)$.

From Proposition 3.11, we have that $(P, \emptyset) \equiv_{\mathbf{C},G} \{\, (\nu P, \nu) \mid \nu \in sat(P, \mathbf{C}, G) \,\}$. By construction, each mapping in $sat(P, \mathbf{C}, G)$ is not comparable to the others. Since $\mathbf{C}, G \not\models Compl(\mu P)$, we have the extension pair $(G, G \cup \widetilde{\mu P})$ as a counterexample for $\mathbf{C}, G \models Compl(P)$.
($\Leftarrow$) By the first claim of Proposition 3.11, we have that μP is saturated wrt. $\mathbf{C}$ and G for each $\mu \in sat(P, \mathbf{C}, G)$. Thus, from the right-hand side of Theorem 3.12 and Lemma 3.10, we have that $\mathbf{C}, G \models Compl(\mu P)$ for each $\mu \in sat(P, \mathbf{C}, G)$. Therefore, we have that $\mathbf{C}, G \models Compl(P)$ by the second claim of Proposition 3.11. □

Example 3.13. Consider our motivating scenario. It is the case that $sat(P_0, \mathbf{C}, G) = \{\, \{\, ?crew \mapsto tony, ?child \mapsto toby \,\} \,\}$. For every mapping μ in $sat(P_0, \mathbf{C}, G)$, it holds that $\widetilde{\mu P_0} \subseteq G$. Thus, by Theorem 3.12 the entailment $\mathbf{C}, G \models Compl(P_0)$ holds.

From looking back at the characterization of completeness entailment in Proposition 3.2, it actually does not give us a concrete way to compute a set of mappings to be used in checking completeness entailment. Now, by Theorem 3.12 it is sufficient for checking completeness entailment to consider only the mappings in $sat(P, \mathbf{C}, G)$ for which we know how to compute.

Simple Practical Optimizations. In what follows we provide two simple optimization techniques of the algorithm: early failure detection and completeness skip. More elaborate optimizations are given in Chapter 4.

(i) Early failure detection. In our algorithm, the containment checks for saturated BGPs are done at the end. Indeed, if there is a single saturated BGP not contained in the graph, we cannot guarantee query completeness (recall Theorem 3.12). Thus, instead of having to collect all saturated BGPs and then check the containment later on, we can improve the performance of the algorithm by performing the containment check right after the saturation check (Line 6 of the algorithm). So, as soon as there is a failure in the containment check, we stop the loop and conclude that the completeness entailment does not hold.

(ii) Completeness skip. Recall the definition of the operator

$$\mathit{epg}((P,\nu),\mathbf{C},G) = \{\, (\mu P, \nu \cup \mu) \mid \mu \in [\![\mathit{cruc}_{\mathbf{C},G}(P)]\!]_G \,\},$$

which relies on the `cruc` operator. Now, suppose that $\mathit{cruc}_{\mathbf{C},G}(P) = P$, implying that we are complete for the whole part of the BGP P. Thus, we actually do not have to instantiate P in the `epg` operator, since we know that the instantiation results will be contained in G anyway due to P's completeness wrt. $\mathbf{C}$ and G. In conclusion, whenever $\mathit{cruc}_{\mathbf{C},G}(P) = P$, we just remove (P,ν) from $\mathbf{P}_{\mathit{working}}$ and thus skip its instantiations.

3.2.3. Complexity

In this subsection, we analyze the complexity of the problem of data-aware completeness entailment. Recall that the complexity of the data-agnostic counterpart is NP-complete (as per Corollary 2.11). The addition of the data graph to the entailment increases the complexity, which is now Π_2^P-complete. The hardness is by reduction from the validity problem of a $\forall\exists$3SAT formula.

Proposition 3.14. *Deciding whether $\mathbf{C}, G \models \mathit{Compl}(P)$ holds, given a set $\mathbf{C}$ of completeness statements, a graph G, and a BGP P, is Π_2^P-complete.*

Proof. The membership proof is as follows. It is the case that $\mathbf{C}, G \not\models \mathit{Compl}(P)$ iff there exists a graph G' containing G where:

- $(G, G') \models \mathbf{C}$, and
- $(G, G') \not\models \mathit{Compl}(P)$.

We guess a mapping μ over P such that $\mu P \nsubseteq G$, which implies that $(G, G \cup \mu P) \not\models \mathit{Compl}(P)$. Then, we check in CoNP that $(G, G \cup \mu P) \models \mathbf{C}$. If it holds, then $\mathbf{C}, G \not\models \mathit{Compl}(P)$ by the counterexample $G' = G \cup \mu P$.

Next, we prove the hardness by reduction from the validity of a ∀∃3SAT formula. The general shape of a formula is as follows:

$$\psi = \forall x_1, \ldots, x_m \exists y_1, \ldots, y_n \ \gamma_1 \land \ldots \land \gamma_k,$$

where each γ_i is a disjunction of three literals over propositions from $\mathit{vars}_\forall \cup \mathit{vars}_\exists$ where $\mathit{vars}_\forall = \{x_1, \ldots, x_m\}$ and $\mathit{vars}_\exists = \{y_1, \ldots, y_n\}$. We will construct a set $\mathbf{C}$ of completeness statements, a graph G, and a BGP P such that the following claim holds:

$$\mathbf{C}, G \models \mathit{Compl}(P) \qquad \text{iff} \qquad \psi \text{ is valid.}$$

Our encoding is inspired by the following approach to check the validity of ψ: Unfold the universally quantified variables $x_1, \ldots, x_m$ in ψ, and then check if for every formula in the set Ψ_{unfold} of the unfolding results, there is an assignment from the existentially quantified variables $y_1, \ldots, y_n$ to make all the clauses evaluate to true.

(Encoding) First, we construct[1]

$$G = \{\, (\mathit{0}, \mathit{varg}, c), (\mathit{1}, \mathit{varg}, c) \,\}$$

and the completeness statement

$$C_\forall = \mathit{Compl}(\{\, (?x, \mathit{varg}, ?y) \,\}),$$

to denote all the assignment possibilities (i.e., 0 and 1) for the universally quantified variables.

Next, we define

$$P_{\mathit{ground}} = \{\, (?x_i, \mathit{varg}, ?c_{x_i}), (?x_i, \mathit{varc}, c_{x_i}) \mid x_i \in \mathit{vars}_\forall \,\}.$$

The idea is that P_{ground} via $C_\forall$ and G will later be instantiated with all possible assignments for the universally quantified variables in ψ.

[1]Recall that we omit namespaces. With namespaces, for example, the 'number' 0 in the encoding can be written as the IRI `http://example.org/0`.

Now, we define

$$P_{neg} = \{\, (\mathit{0}, neg, \mathit{1}), (\mathit{1}, neg, \mathit{0}) \,\},$$

which says that 0 is the negation of 1, and vice versa. This BGP is used later on to assign values for all the propositional variables and their negations. Then, we define

$$P_{true} = \{\, (\mathit{1}, \mathit{1}, \mathit{1}), \ldots, (\mathit{0}, \mathit{0}, \mathit{1}) \,\},$$

to denote the seven possible valid values for a clause. Our BGP P we want to check for completeness is therefore as follows:

$$P = P_{true} \cup P_{neg} \cup P_{ground}.$$

Now, we want to encode the structure of the formula ψ. For each propositional variable p_i, we encode the positive literal p_i as the variable $var(p_i) = ?p_i$ and the negative literal $\neg p_i$ as the variable $var(\neg p_i) = ?\neg p_i$. Given a clause $\gamma_i = l_{i1} \vee l_{i2} \vee l_{i3}$, the operator $tp(\gamma_i)$ maps γ_i to a triple pattern $(var(l_{i1}), var(l_{i2}), var(l_{i3}))$. We then define the following BGP to encode the structure of ψ:

$$P_\psi = \{\, tp(\gamma_i) \mid \gamma_i \text{ occurring in } \psi \,\}.$$

To encode the inverse relationship between a positive literal and a negative literal, we use the following:

$$P_{poss} = \{\, (?p_i, neg, ?\neg p_i), (?\neg p_i, neg, ?p_i) \mid p_i \in vars_\forall \cup vars_\exists \}.$$

This pattern will later be instantiated accordingly wrt. P_{neg}. Now, for capturing the assignments of the universally quantified variables in P, we use

$$P_\forall = \{\, (?x_i, varc, c_{x_i}) \mid x_i \in vars_\forall \,\}.$$

We are now ready to construct the following completeness statement:

$$C_\psi = Compl(P_{true} \cup P_{poss} \cup P_\forall \cup P_\psi).$$

In summary, our encoding consists of the following ingredients: the set $\mathbf{C} = \{\, C_\forall, C_\psi \,\}$ of completeness statements, the graph G, and the BGP P. Let us now prove the claim mentioned above.

(Proof for Encoding) Recall the approach we mentioned above to check the validity of the formula ψ. To simulate the unfolding of the universally quantified variables, we rely on the equivalent partial grounding operator $\mathit{epg}((P,\emptyset),\mathbf{C},G)$ as in Algorithm 1 which involves the *cruc* operator. Accordingly, $\mathit{cruc}_{\mathbf{C},G}(P) = P \cap \tilde{id}^{-1}(T_{\mathbf{C}}(\tilde{P} \cup G))$ by definition. By construction, the statement $C_\forall$ captures the $(?x_i, \mathit{varg}, ?c_{x_i})$ part of the BGP P where $x_i \in \mathit{vars}_\forall$. Thus, by the construction of G, it is the case that $\mathit{epg}((P,\emptyset),\mathbf{C},G)$ consists of 2^m partially mapped BGPs, where m is the number of the universally quantified variables in ψ. Each of the partially mapped BGPs corresponds to an assignment for the universally quantified variables in the set Ψ_{unfold} of the unfolding results of ψ.

Now, we prove the simulation of the next step, the existential checking. For each partially mapped BGP $(\mu P, \mu)$ in the unfolding results $\mathit{epg}((P,\emptyset),\mathbf{C},G)$, it is either $\mathit{epg}((\mu P,\mu),\mathbf{C},G) = \emptyset$ or $\mathit{epg}((\mu P,\mu),\mathbf{C},G) = \{ (\mu P,\mu) \}$. Let us see what this means.

By construction, the former case happens whenever $T_{\mathbf{C}}(\widetilde{\mu P} \cup G) = \widetilde{\mu P}$ holds, from the fact that $[\![\mu P]\!]_G = \emptyset$. Furthermore, it is the case that $T_{\mathbf{C}}(\widetilde{\mu P} \cup G) = \widetilde{\mu P}$ iff there is a mapping ν from the encoding $?y_i$ of the existentially quantified variables in P_ψ such that $\nu(\mu P_\psi) \subseteq P_{\mathit{true}}$. Note that the mapping ν simulates a satisfying assignment for the corresponding existentially quantified formula in the set Ψ_{unfold}. Whenever this holds for all $(\mu P,\mu) \in \mathit{epg}((P,\emptyset),\mathbf{C},G)$, from Proposition 3.8 we can conclude that $(P,\emptyset) \equiv_{\mathbf{C},G} \emptyset$, and therefore $\mathbf{C}, G \models \mathit{Compl}(P)$. Also, because we have the satisfying assignments for all the corresponding existentially quantified formulas in the set Ψ_{unfold}, the formula ψ evaluates to true.

The latter case happens whenever $T_{\mathbf{C}}(\widetilde{\mu P} \cup G) \neq \widetilde{\mu P}$, since there is no mapping ν from the encoding $?y_i$ of the existentially quantified variables in P_ψ such that $\nu(\mu P_\psi) \subseteq P_{\mathit{true}}$. This simulates the failure in finding a satisfying assignment for the corresponding existentially quantified formula in the set Ψ_{unfold}. This implies that ψ evaluates to false. However, whenever the latter case happens, it means that $(\mu P,\mu)$ is saturated. By construction, it is the case $\widetilde{\mu P} \nsubseteq G$. From Lemma 3.10 and Proposition 3.8, we conclude that $\mathbf{C}, G \not\models \mathit{Compl}(P)$. □

One might wonder, if some parts of the inputs were fixed, what would be the complexity of the entailment problem. We answer this question in the following series of propositions.

First, let us fix the input graph G. This does not change the complexity, that is, the problem is still Π_2^P-complete. The reason is that, the reduction from the validity problem of a ∀∃3SAT formula can be done even with a fixed graph.

Proposition 3.15. *Deciding whether* $\mathbf{C}, G \models \mathit{Compl}(P)$ *holds, given a set* $\mathbf{C}$ *of completeness statements, a fixed graph* G*, and a BGP* P*, is* Π_2^P*-complete.*

Proof. The membership follows immediately from Proposition 3.14, while the hardness follows from the reduction proof of that proposition, in which the graph is fixed. □

Now, we want to see the complexity when the BGP P is fixed. Recall that in the algorithm, P dominates the complexity of the instantiation process in the *epg* operator. When it is fixed, the size of the instantiations is bounded polynomially, reducing the complexity of the entailment problem to NP-complete. Note it is still NP-hard even when the input graph G is fixed.

Proposition 3.16. *Deciding whether* $\mathbf{C}, G \models \mathit{Compl}(P)$ *holds, given a set* $\mathbf{C}$ *of completeness statements, a graph* G*, and a fixed BGP* P*, is NP-complete.*

Proof. The membership relies on Algorithm 1 and Theorem 3.12. Recall that the algorithm contains the *epg* operator, which performs grounding based on the crucial part over the graph G. However, now since the BGP is fixed, the size of the grounding results is therefore bounded polynomially. Consequently, the only source of complexity is from the finding of the crucial part of BGPs, which can be done in NP (note that here the completeness statements are not fixed).

The hardness follows immediately from Proposition 3.17. □

Proposition 3.17. *Deciding whether* $\mathbf{C}, G \models \mathit{Compl}(P)$ *holds, given a set* $\mathbf{C}$ *of completeness statements, a fixed graph* G*, and a fixed BGP* P*, is NP-complete.*

Proof. The membership follows immediately from Proposition 3.16.

The proof for NP-hardness is by means of reduction from the 3-colorability problem of directed graphs, which is known to be NP-hard [42]. We encode the problem graph $G_p = (V, E)$, i.e., the directed graph we want to check whether it is 3-colorable, as a set

$\mathit{triples}(G_p)$ of triple patterns. We associate to each vertex $v \in V$, a new variable $?v$. Then, we define $\mathit{triples}(G_p)$ as the union of all triple patterns $(?s, \mathit{edge}, ?o)$ created from each pair $(s, o) \in E$ where $?s$ is the associated variable of s, edge is an IRI and $?o$ is the associated variable of o. Let the BGP P_{col} be:

$$\{(r, \mathit{edge}, g), (r, \mathit{edge}, b), (g, \mathit{edge}, r), (g, \mathit{edge}, b), \\ (b, \mathit{edge}, r), (b, \mathit{edge}, g)\}$$

Next, we create the following completeness statement C_p:

$$\mathit{Compl}(\mathit{triples}(G_p) \cup P_{\mathit{col}})$$

Let G be the empty set. Thus, the following claim holds:

$$\text{The problem graph } G_p \text{ is 3-colorable} \quad \text{if and only if} \\ \{ C_p \}, G \models \mathit{Compl}(P_{\mathit{col}})$$

Proof of the claim: "$\Rightarrow$" Assume G_p is 3-colorable. Thus, there must be a mapping μ from all the vertices in G_p to an element from the set $\{ r, g, b \}$ such that no adjacent nodes have the same color. This mapping can then be reused for mapping the `CONSTRUCT` query of the statement C_p to the frozen version of the BGP P_{col}, which then ensures the completeness of P_{col}.

"$\Leftarrow$" We will prove by contrapositive. Assume that G_p is not 3-colorable. Thus, there is no mapping from the vertices in G_p to an element from the set $\{ r, g, b \}$ such that any adjacent node has a different color. Suppose that there is an extension pair (G, G') such that G' is the color graph $\{ (r, \mathit{edge}, g), \ldots, (b, \mathit{edge}, g) \}$. From the construction of C_p, it is the case that $(G, G') \models \{ C_p \}$ but $[\![P_{\mathit{col}}]\!]_G \neq [\![P_{\mathit{col}}]\!]_{G'}$. Thus, $\{ C_p \}, G \not\models \mathit{Compl}(P_{\mathit{col}})$. □

Let us now see the complexity when the set of statements $\mathbf{C}$ is fixed. In the algorithm, $\mathbf{C}$ dominates the complexity of the $T_{\mathbf{C}}$ operator used in computing the crucial part. When it is fixed, the $T_{\mathbf{C}}$ operator can be done in PTIME, reducing the complexity of the entailment problem to CoNP-complete. Again, fixing also the graph does not change the complexity.

Proposition 3.18. *Deciding whether $\mathbf{C}, G \models \mathit{Compl}(P)$ holds, given a fixed set $\mathbf{C}$ of completeness statements, a graph G, and a BGP P, is CoNP-complete.*

Proof. The membership proof is as follows. It is the case that $\mathbf{C}, G \not\models$ $Compl(P)$ iff there exists a graph G' containing G where:

- $(G, G') \models \mathbf{C}$, and
- $(G, G') \not\models Compl(P)$.

We guess a mapping μ over P such that $\mu P \not\subseteq G$, which implies that $(G, G \cup \mu P) \not\models Compl(P)$. Then, we check in PTIME (since $\mathbf{C}$ is now fixed) the entailment $(G, G \cup \mu P) \models \mathbf{C}$. If it holds, then $\mathbf{C}, G \not\models Compl(P)$ by the counterexample $G' = G \cup \mu P$.

The hardness follows immediately from Proposition 3.19. □

Proposition 3.19. *Deciding whether* $\mathbf{C}, G \models Compl(P)$ *holds, given a fixed set* $\mathbf{C}$ *of completeness statements, a fixed graph* G*, and a BGP* P*, is CoNP-complete.*

Proof. The membership follows immediately from Proposition 3.18.

The proof for CoNP-hardness is by means of reduction from the 3-incolorability problem of directed graphs. We encode the problem graph $G_p = (V, E)$, i.e., the directed graph we want to check whether it is 3-incolorable, as a set $triples(G_p)$ of triple patterns. We associate to each vertex $v \in V$, a new variable $?v$. Then, we define $triples(G_p)$ as the union of all triple patterns $(?s, edge, ?o)$ created from each pair $(s, o) \in E$ where $?s$ is the associated variable of s, $edge$ is an IRI and $?o$ is the associated variable of o. Let the BGP P be:

$$triples(G_p) \cup \{\, (a, b, c) \,\}$$

Let the graph G be the color graph:

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

Next, we create the following completeness statement C:

$$Compl((?x, edge, ?y))$$

Thus, the following claim holds:

The problem graph G_p is 3-incolorable if and only if $\{\, C \,\}, G \models Compl(P)$.

Proof of the claim: "⇒" The proof relies on Algorithm 1 and Theorem 3.12. Assume G_p is 3-incolorable. By construction, the part $triples(G_p)$ of the BGP P can be grounded completely due to the statement C, that is, the crucial part operator *cruc* returns exactly that part. However, as G_p is 3-incolorable, there is no mapping μ from all the vertices in G_p to an element from the set $\{ r, g, b \}$ such that no adjacent nodes have the same color. Thus, the *epg* operator returns the empty set as evaluating *triples*(G_p) over G yields the empty answer. This means that the grounding does not output any BGP that needs to be checked anymore for its completeness. Hence, it is the case that $\{ C \}, G \models \mathit{Compl}(P)$.

"$\Leftarrow$" We will prove by contrapositive. Assume that G_p is 3-colorable. Thus, there must be a mapping μ from all the vertices in G_p to an element from the set $\{ r, g, b \}$ such that no adjacent nodes have the same color. Take such a mapping μ arbitrarily. By construction, the part *triples*(G_p) of the BGP P can be grounded completely due to the statement C, that is, the crucial part operator *cruc* returns exactly that part. Since the graph G_p is 3-colorable, we can then reuse the mapping μ for mapping *triples*(G_p) to G. The *epg* operator results therefore include that mapping, which is then applied to the remaining part of P, that is, the triple pattern (a, b, c). Note that the triple pattern consists of only constants, so the mapping application has no effect. Now we have to check the completeness of (a, b, c). As no completeness statements can be evaluated over that remaining part, it is then the case that we are already saturated for (a, b, c). By Theorem 3.12, the BGP P can be guaranteed to be complete iff all saturated instantiations wrt. $\{ C \}$ are in G. However, clearly (a, b, c) is not in G. Thus, we have that $\{ C \}, G \not\models \mathit{Compl}(P)$. □

Finally, the following proposition tells us that fixing both the set of statements $\mathbf{C}$ and the BGP P reduces the complexity to PTIME.

Proposition 3.20. *Deciding whether* $\mathbf{C}, G \models \mathit{Compl}(P)$ *holds, given a fixed set* $\mathbf{C}$ *of completeness statements, a graph* G*, and a fixed BGP* P*, is in PTIME.*

Proof. The proof relies on Algorithm 1 and Theorem 3.12. Recall that the algorithm contains the *epg* operator, which performs grounding based on the crucial part over the graph G. However, now since the BGP is fixed, the size of the grounding results is therefore bounded polynomially. Moreover, now that the completeness statements are fixed, the finding of the crucial part can then be done in PTIME. Hence, the overall procedure can be done in PTIME. □

This result corresponds to some practical cases when queries are assumed to be of limited length[2] and hence, so are completeness statements (which are essentially also queries).

input			*complexity*
C	*G*	*P*	
✓	✓	✓	Π_2^P-complete
✓	×	✓	Π_2^P-complete
✓	✓	×	NP-complete
✓	×	×	NP-complete
×	✓	✓	CoNP-complete
×	×	✓	CoNP-complete
×	✓	×	in PTIME

Table 3.1. Complexity table for the data-aware completeness entailment problem with various input fixes (× denotes 'fixed')

Our complexity results with various input fixes can be summarized in Table 3.1. From this complexity study, it is therefore of our interest to study how well the problem of completeness entailment for both the data-agnostic and data-aware cases may behave in practice. We will later provide optimization techniques, as well as experimental evaluations of the problem in Chapter 4.

3.3. SP-statements

In the previous section, we have provided completeness characterizations for queries by using generic completeness statements. Yet, in some practical cases a simpler fragment of completeness statements might be sufficient for the task at hand. This section identifies SP-statements, a fragment of completeness statements having several properties that are suitable for RDF data sources with the entity-centric, crowdsourced basis.

3.3.1. Motivation

An *SP-statement* $Compl((s, p, ?v))$ is a completeness statement with only one triple pattern in the statement's BGP, where the subject and

[2]as also customary in the database theory when analyzing the data complexity of query evaluation

the predicate are IRIs, and the object is a variable. In our motivating scenario (see Section 3.1), all the completeness statements are in fact SP-statements. The statements possess the following properties, which make them suitable for practical use:

- Having a simple structure, completeness statements within this fragment are easy to create and to be read. Thus, they are suitable for *crowdsourced* KBs, where humans are involved.
- An SP-statement denotes the completeness of all the property values of the entity which is the subject of the statement. This fits *entity-centric* KBs like Wikidata, where data is organized according to entities (i.e., each entity has its own data page).
- Despite their simplicity, SP-statements can be used to guarantee the completeness of more complex queries such as queries whose length is greater than one (as illustrated by our motivating scenario).

3.3.2. SP-Indexing

We describe here how to optimize data-aware completeness entailment check with SP-statements. Recall our generic algorithm to check completeness entailment:
In the *cruc* operator within the *epg* operator (Line 5 of Algorithm 1), we have to compute $T_{\mathbf{C}}(\tilde{P} \cup G)$, that is, evaluate all `CONSTRUCT` queries of the completeness statements in $\mathbf{C}$ over the graph $\tilde{P} \cup G$. This may be problematic if there are a large number of completeness statements in $\mathbf{C}$. Thus, we want to avoid such costly $T_{\mathbf{C}}$ applications. Given that completeness statements are SP-statements, we may instead look for the statements having the same subject and predicate of the triple patterns in the BGP. The crucial part of the BGP P wrt. $\mathbf{C}$ and G are the triple patterns for which there is an SP-statement with a matching subject and predicate.

Proposition 3.21. *Given a BGP P, a graph G, and a set $\mathbf{C}$ of SP-statements, it is the case that $cruc_{\mathbf{C},G}(P) = \{ (s,p,o) \in P \mid$ there exists a statement $Compl(\{ (s,p,?v) \}) \in \mathbf{C} \}$.*

From the above proposition, to get the crucial part, we only have to find an SP-statement with the same subject and predicate for each triple pattern of the BGP, and thus, the graph G does not play any role. In practice, we can facilitate this search via a standard hashmap, providing constant-time performance, also for other basic operations such as `add` and `delete`. The hashmap provides a mapping from the concatenation of the subject and the predicate of a statement to the statement itself. To illustrate, the hashmap of the completeness statements in our motivating scenario is as follows: $\{\, \textit{a99-crew} \mapsto C_1, \textit{tony-child} \mapsto C_2, \textit{ted-child} \mapsto C_3 \,\}$.

Complexity-wise, it is the case that when completeness statements are only of 1 triple pattern (i.e., a close generalization of SP-statements), the problem of data-aware completeness entailment is CoNP-complete. This is in contrast to the complexity for general cases, which is Π_2^P-complete (as in Proposition 3.14).

Proposition 3.22. *Deciding whether $\mathbf{C}, G \models \mathit{Compl}(P)$ holds, given a set $\mathbf{C}$ of completeness statements of length 1, a graph G, and a BGP P, is CoNP-complete.*

Proof. The Co-NP membership proof is as follows. It is the case that $\mathbf{C}, G \not\models \mathit{Compl}(P)$ iff there exists a graph G' containing G where:

- $(G, G') \models \mathbf{C}$, and
- $(G, G') \not\models \mathit{Compl}(P)$.

We guess a mapping μ over P such that $\mu P \not\subseteq G$, implying that $(G, G \cup \mu P) \not\models \mathit{Compl}(P)$. Then, we check $(G, G \cup \mu P) \models \mathbf{C}$, which can now be done in PTIME since completeness statements are of length 1. If it holds, then $\mathbf{C}, G \not\models \mathit{Compl}(P)$ by the counterexample $G' = G \cup \mu P$.

The hardness proof is by reduction from the problem of graph 3-incolorability. We refer to the CoNP hardness proof of Proposition 3.19, in which the only completeness statement used is also of length 1.

□

3.3.3. Experimental Evaluation

Now that we have an indexing technique for SP-statements, we want to investigate the performance of completeness checking using such statements. To do so, we perform an experimental evaluation with a realistic scenario, where we compare the runtime of completeness entailment when query completeness can be guaranteed (i.e., the success case), completeness entailment when query completeness cannot be guaranteed (i.e., the failure case), and query evaluation.

Experimental Setup. Our reasoning algorithm and indexing modules were implemented in Java using the Apache Jena library.[3] We used Jena-TDB as the triple store of our experiment. The SP-indexing was implemented using the standard Java `hashmap`, where the keys are strings constructed from the concatenation of the subject and predicate of completeness statements, and the values are Java objects representing completeness statements. All experiments were done on a standard laptop with a 2.4 GHz Intel Core i5 and 8 GB of memory.

There were three ingredients for the experiment: a graph, completeness statements, and queries. For the graph, we used the direct-statement fragment of the Wikidata graph, which does not include qualifiers nor references (that is, only property-value pairs of entities) and consists of 100 mio triples.[4] For the queries and completeness statements, we want to have a variety in the selectivity. Therefore, we chose the following query templates (or pattern queries) over Wikidata, which later will be used to generate the queries and statements:

1. Give all mothers (= P25) of mothers of mothers.
   $P_1 = \{ (?v, \mathit{P25}, ?w), (?w, \mathit{P25}, ?x), (?x, \mathit{P25}, ?y) \}$
2. Give the crew (= P1029) of a thing, the astronaut missions (= P450) of each such crew, and the operator (= P137) of the missions.
   $P_2 = \{ (?v, \mathit{P1029}, ?w), (?w, \mathit{P450}, ?x), (?x, \mathit{P137}, ?y) \}$
3. Give the administrative divisions (= P150) of a thing, the administrative divisions of those divisions, and their area (= P2046).
   $P_3 = \{ (?v, \mathit{P150}, ?w), (?w, \mathit{P150}, ?x), (?x, \mathit{P2046}, ?y) \}$

Let us describe how we generate the queries and completeness statements. To generate queries, we simply evaluated each pattern query over the graph, and instantiated the variable $?v$ of each pattern query with the corresponding mappings from the evaluation. We recorded 5,200 queries instantiated from P_1, 57 queries from P_2, and 475 queries from P_3. Each pattern query had a different average number of query results: the instantiations of P_1 gave 1 result, those of P_2 gave 4 results, and those of P_3 gave 108 results on average. So, we had a variety of query selectivity.

[3]`https://jena.apache.org/`

[4]`https://tools.wmflabs.org/wikidata-exports/rdf/exports/20151130/`

To generate completeness statements, from each generated query, we iteratively evaluated each triple pattern from left to right, and constructed SP-statements from the instantiated subject and the predicate of the triple patterns. This way, we guaranteed that all the queries can be answered completely. We generated in total around 1.7 mio statements, with 30,072 statements for P_1, 484 statements for P_2, and 1,682,263 statements for P_3. Such a large number of completeness statements would make completeness checks without indexing very slow: Performing just a single application of the $T_\mathbf{C}$ operator with all these statements, which occurs inside the `cruc` operator of the algorithm took around 20 minutes without SP-indexing. Note that in a completeness check, there might be many $T_\mathbf{C}$ applications.

Now we describe how to observe the behavior when queries cannot be guaranteed to be complete, that is, the failure case. In this case, we dropped randomly 20% of the completeness statements for each pattern query. To make up the statements we dropped, we added dummy statements with the number equal to the number of dropped statements. This way, we ensured the same number of completeness statements for both the success and failure case.

For each query pattern, we measured the runtime of completeness checking for both the success case and the failure case, and then query evaluation for the success case.[5] We took 40 sample queries for each pattern query, repeated each run 10 times, and reported the median of these runs.

Experimental Results. The experimental results are shown in Figure 3.1. Note that the runtime is in log scale. We can see that in all cases, the runtime increases with the first pattern query having the lowest runtime, and the third pattern query having the highest runtime. This is likely due to the increased number of query results. We observe that in all pattern queries, the completeness check when queries are guaranteed to be complete is slower than when completeness cannot be guaranteed. We suspect that this is because in the former case, variable instantiations have to be performed much more often than in the latter case (that is, it has to generate all possible instantiations). In the latter case, as soon as we find a saturated BGP not contained in the graph, we stop the loop in the algorithm

[5]We did not measure query evaluation time for failure case since query evaluation is independent of the completeness of the query.

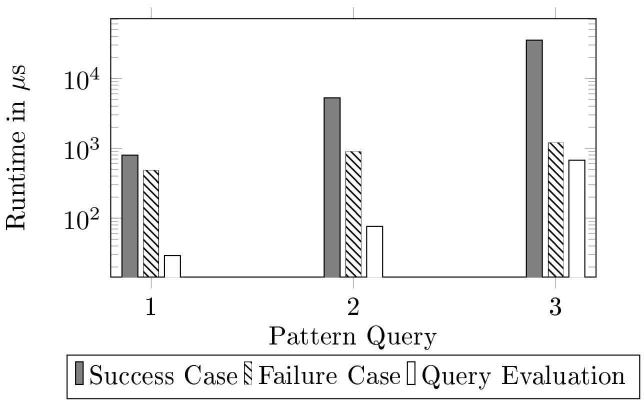


Figure 3.1. Experiment Results of Completeness Entailment

and return `false`, meaning that the query completeness cannot be guaranteed. For queries with large results, such a termination might be done much earlier than when the queries are complete. This possibly explains the increasing runtime gap between the success case and failure case in the figure.

In absolute scale, the completeness check runs relatively fast, with 796 μs for P_1, 5,264 μs for P_2, and 35,130 μs for P_3 in success case; and 485 μs for P_1, 903 μs for P_2, and 1,209 μs for P_3 in failure case. Note that as mentioned before, completeness checking without indexing is not feasible at all here, as there are a large number of completeness statements, making the T_C application very slow (i.e., 20 minutes for a single application). For all pattern queries, however, query evaluation runs faster than completeness checking. This is because completeness checking may involve several query evaluations during the instantiation process with the `epg` operator.

To conclude, we have observed that completeness checking with a large number of SP-statements can be done reasonably fast, even for large datasets, by employing indexing. Also, we observe a clear positive correlation between the number of query results and the runtime of completeness checking. Last, performing a completeness check when a query is complete is slower than that when a query cannot be guaranteed to be complete.

3.4. No-value Statements

In this section, we focus on the problem of non-existent information: stating that some parts of data do not exist in the real world. Non-existent information is related to data completeness in the following way: if we know that some parts of data do not exist, then any data source is trivially complete for those parts. We introduce no-value statements, a fragment of completeness statements that is suited to expressing the non-existence of information in RDF. With no-value statements, the problem of checking query completeness (i.e., query answers are complete?) is now shifted to the problem of checking query emptiness (i.e., query answer is truly empty?). We first motivate no-value statements, then provide a formal characterization of query emptiness entailment with no-value statements, and describe how one can concretely represent no-value statements in RDF.

3.4.1. Motivation

RDF is mainly used to express positive information. However, representing negative information is often of interest in practice. For instance, Wikidata [111] has the following information about Elizabeth I not having any children.[6]

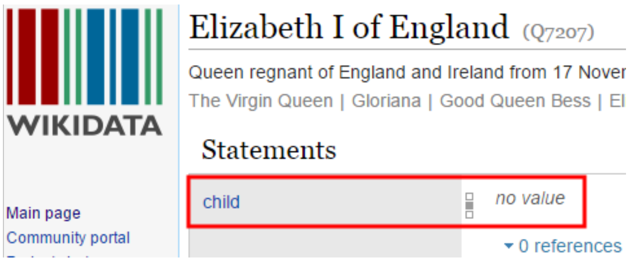

Elizabeth I of England (Q7207)
Queen regnant of England and Ireland from 17 Nover
The Virgin Queen | Gloriana | Good Queen Bess | El
Statements
child no value

Figure 3.2. No-value information on Wikidata

In the above figure, Wikidata explicitly states that Elizabeth I had no children since the property `child` has “no value”.[7] This is different than not recording anything at all which would imply possibly incomplete information for the children of Elizabeth I. To express

[6]`https://www.wikidata.org/wiki/Q7207`

[7]For further information about no values on Wikidata, refer to `https://www.wikidata.org/wiki/Wikidata:Glossary`.

this in RDF, one may be tempted to assign a special datatype constant `noValue` to represent the no-value information of the children of Elizabeth I, creating the triple $(elizabethI, child, noValue)$. However, this creates a problem since executing the SPARQL `ASK` query $Q = (\{\,\}, \{(elizabethI, child, ?y)\})$ asking if Elizabeth I has a child, would give the answer 'yes'. Indeed, due to no formal definition, it is not clear how to properly use `noValue`.

The notion of no-value information was first introduced in relational databases [3]. There, the term 'null value' was used, which may have different meanings: there exists no value (i.e., non-existence); there exists a value but it is unknown; or it is unknown whether a value exists. For the second case, we can leverage RDF blank nodes, whereas for the third case, the open-world assumption (OWA) of RDF simply permits it. However, RDF cannot represent the first case, which is the one of no-value nulls, while in fact this no-value information is useful to distinguish this case from incomplete information. Furthermore, by having no-value information, an empty query answer can have two different meanings: it may be empty because of possibly incomplete information, or it may be truly empty because such information does not exist in the real world. From the practical side, Wikidata itself contains in total about 19,000 pieces of no-value information over 269 properties.[8] Given such amount, it is therefore potentially beneficial (e.g., for checking SPARQL query emptiness) if no-value information can be formalized and represented in RDF in a standardized way.

3.4.2. Formalization

Let us formalize no-value information. We first define no-value statements to capture which information is non-existent. Such statements denote that a particular concept cannot exist wrt. the real world.

Definition 3.23 (No-Value Statement)**.** A *no-value statement* N is defined as $No(P)$ where P is a BGP. To N, we associate the `CONSTRUCT` query $Q_N = (P, P)$.

We use BGPs to have a flexibility to represent complex no-values which need more than one triple pattern. For example, one can state

[8] as per Feb 18, 2017

that "Elizabeth I has no child" with $N_{el} = No((elizabethI, child, ?c))$, whereas "Obama has no son" with

$$N_{ob} = No((obama, child, ?c), (?c, gender, male)).$$

Now, we want to give the semantics of no-value statements. As before, we use an extension pair to model the OWA of RDF graphs. Having no-value statements restricts the possibilities of extension pairs since they must not contain any instantiation of the information denoted by the statements. Over a graph G, we define the *transfer operator* $T_{\mathcal{N}}(G) = \bigcup_{N \in \mathcal{N}} [\![Q_N]\!]_G$. We define the semantics of no-value statements as follows.

Definition 3.24 (Satisfaction of No-Value Statements)**.** An extension pair (G, G') satisfies a set $\mathcal{N}$ of no-value statements, written as $(G, G') \models \mathcal{N}$, if and only if $T_{\mathcal{N}}(G') = \emptyset$.

Note that since $G \subseteq G'$ holds by the definition of an extension pair, $T_{\mathcal{N}}(G') = \emptyset$ implies $T_{\mathcal{N}}(G) = \emptyset$. Next, we define the emptiness of a query over an extension pair.

Definition 3.25 (Query Emptiness)**.** Let (G, G') be an extension pair and Q a query. To express that Q *is empty*, we write $Empty(Q)$. It is the case that $(G, G') \models Empty(Q)$ if and only if $[\![Q]\!]_{G'} = \emptyset$.

Query emptiness over one extension pair does not mean that it always holds also over other extension pairs. For this reason, we define the query emptiness entailment: that $\mathcal{N} \models Empty(Q)$ holds, if for any extension pair $(G, G') \models \mathcal{N}$, we have that $(G, G') \models Empty(Q)$. If the entailment holds, we can guarantee that the query will always return an empty answer no matter which possible extensions of a graph are considered. The next theorem characterizes query emptiness entailment: whenever there is some part of the query that cannot return any answer due to no-value information, then the whole query does not return any answer. Via this theorem, we are able to distinguish between empty query answers from possibly incomplete information, and empty query answers from non-existent information.

Theorem 3.26 (Query Emptiness Entailment)**.** *Let $\mathcal{N}$ be a set of no-value statements, Q be a query, and $\tilde{P}$ be the prototypical graph of Q. It is the case that $\mathcal{N} \models Empty(Q)$ if and only if $T_{\mathcal{N}}(\tilde{P}) \neq \emptyset$.*

Proof. ($\Rightarrow$) We will prove by contrapositive. Assume $T_{\mathcal{N}}(\tilde{P}) = \emptyset$. We will show that $\mathcal{N} \not\models \mathit{Empty}(Q)$. Take the extension pair $(\emptyset, \tilde{P})$. By the assumption, it is the case that $(\emptyset, \tilde{P}) \models \mathcal{N}$. However, we have that $(\emptyset, \tilde{P}) \not\models \mathit{Empty}(Q)$ since $[\![Q]\!]_{\tilde{P}}$ is not empty by the definition of the prototypical graph $\tilde{P}$.
($\Leftarrow$) Assume $T_{\mathcal{N}}(\tilde{P}) \neq \emptyset$. Take any extension pair (G, G') such that $(G, G') \models \mathcal{N}$. We will show that $(G, G') \models \mathit{Empty}(Q)$. It is sufficient to show that $[\![Q]\!]_{G'} = \emptyset$. There must be a no-value statement $N \in \mathcal{N}$ for a witness of our assumption that $T_{\mathcal{N}}(\tilde{P}) \neq \emptyset$. Thus, we have that $\emptyset \neq [\![Q_N]\!]_{\tilde{P}} \subseteq \tilde{P}$.

As $(G, G') \models \mathcal{N}$, it must be the case that $[\![Q_N]\!]_{G'} = \emptyset$. Assume that $[\![Q]\!]_{G'} \neq \emptyset$. Thus, there must be a mapping $\mu \in [\![P]\!]_{G'}$. This implies that $\mu(P) \subseteq G'$. Thus, it is the case that $\emptyset \neq [\![Q_N]\!]_{\mu\tilde{id}^{-1}\tilde{P}} \subseteq \mu\tilde{id}^{-1}\tilde{P}$. Since $\mu\tilde{id}^{-1}\tilde{P} = \mu P \subseteq G'$, we have that $\emptyset \neq [\![Q_N]\!]_{G'}$, which contradicts our assumption that $(G, G') \models \mathcal{N}$. Thus, $[\![Q]\!]_{G'} = \emptyset$ holds. □

The complexity of the problem of query emptiness entailment is NP-complete, in contrast to the complexity for general cases of data-aware completeness entailment, which is Π_2^P-complete (as in Proposition 3.14).

Proposition 3.27. *Deciding whether the entailment $\mathcal{N} \models \mathit{Empty}(Q)$ holds, given a set $\mathcal{N}$ of no-value statements and a query Q, is NP-complete.*

Proof. The NP membership is by means of Theorem 3.26. As stated there, it is the case that $\mathcal{N} \models \mathit{Empty}(Q)$ iff $T_{\mathcal{N}}(\tilde{P}) \neq \emptyset$ where $\tilde{P}$ is the prototypical graph of Q. By definition, $T_{\mathcal{N}}(\tilde{P}) \neq \emptyset$ iff there is a no-value statement $N = \mathit{No}(P_N)$ in $\mathcal{N}$ such that $[\![Q_N]\!]_{\tilde{P}}$ is not empty, that is, $[\![Q_N]\!]_{\tilde{P}}$ contains a mapping over $\mathit{var}(P_N)$, say, μ, such that $\mu P_N \subseteq \tilde{P}$. The NP entailment check can thus be done as follows: We guess such a no-value statement N and a mapping μ, and then verify in PTIME that $\mu P_N \subseteq \tilde{P}$.

The NP hardness is by reduction from graph 3-colorability problem, known to be NP-hard [42]. We encode the problem graph $G_p = (V, E)$, i.e., the directed graph we want to check whether it is 3-colorable, as the set $\mathit{triples}(G_p)$ of triple patterns. We associate to each vertex $v \in V$, a new variable $?v$. Then, we define $\mathit{triples}(G_p)$ as the union of all triple patterns $(?s, \mathit{edge}, ?o)$ created from each pair

$(s, o) \in E$ where $?s$ is the associated variable of s, *edge* is an IRI and $?o$ is the associated variable of o. Let the BGP P_{col} be:

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

Next, we create the following no-value statement N_p:

$$No(triples(G_p) \cup P_{col})$$

The following claim holds:

The problem graph G_p is 3-colorable if and only if $\{ N_p \} \models Empty((\{\}, P_{col}))$.

Proof of the claim: "$\Rightarrow$" When the problem graph G_p is 3-colorable, we can therefore reuse the color mapping from G_p to the 3 colors, in the mapping from the `CONSTRUCT` query of N_P to $\tilde{P}_{col}$, which is a witness of $T_{\{ N_p \}}(\tilde{P}_{col}) \neq \emptyset$ (recall Theorem 3.26).
"$\Leftarrow$" When the problem graph G_p is 3-incolorable, there is no color mapping from G_p to the 3 colors. By construction of N_p, it is the case that $T_{\{ N_p \}}(\tilde{P}_{col}) = \emptyset$, implying $\{ N_p \} \not\models Empty((\{\}, P_{col}))$. □

Example 3.28. Consider the no-value statement

$$N_{ob} = No((obama, child, ?c), (?c, gender, male))$$

as above and the query Q_{sch} below,

$$(\{?c, ?s\}, \{ (obama, child, ?c), (?c, gender, male), (?c, school, ?s) \}),$$

asking for the schools of Obama's sons. We have that $T_{\{N_{ob}\}}(\tilde{P}_{sch}) \neq \emptyset$. Thus, from Theorem 3.26, it holds that $\{N_{ob}\} \models Empty(Q_{sch})$. This means that Q_{sch} returns the empty answer because of the non-existence of the asked information, not by the incompleteness of the data source. In contrast, suppose the constant `male` in the query Q_{sch} were the variable `?g`. If Q_{sch} returns the empty answer over the data source, that may be due to the incompleteness of the data source.

RDF Representation of No-value Statements. To increase the potential practical benefits of our no-value formalization, no-value statements should be able to be represented in RDF. Such a representation provides a structured and standardized way of processing no-value statements. The representation of no-value statements follows a similar fashion as completeness statements (see Section 2.2). Given a no-value statement

$$No((s_1, p_1, o_1), \ldots, (s_n, p_n, o_n)),$$

we represent the statement as a resource of the class `NoValStatement`, while we represent each triple pattern in the similar way as triple patterns in completeness statements. The no-value vocabulary is available at `http://completeness.inf.unibz.it/no-value`. For instance, we represent the no-value statement "Obama has no sons" as follows:[9]

```
ex:sonsOfObama a no:NoValStatement ;
  rdfs:comment "A no-value statement of Obama having no sons."@en ;
  no:hasPattern [ no:subject dbp:Barack_Obama ;
    no:predicate dbo:child ;
    no:object [no:varName "c"] ] ;
  no:hasPattern [ no:subject [no:varName "c"] ;
    no:predicate dbo:gender ;
    no:object dbp:Male] .
```

3.5. Related Work

Data completeness concerns the breadth, depth, and scope of information [112]. In the area of relational databases, Motro [80] and Levy [64] were among the first to investigate data completeness. Motro developed a sound technique to check query completeness based on database views, while Levy introduced the notion of local completeness statements to denote which parts of a database are complete. Razniewski and Nutt [96] further extended their results by reducing completeness reasoning to containment checking, for which many algorithms are known, and characterizing the complexity of reasoning for different classes of queries. In terms of their terminology, our completeness entailment problem is one of QC-QC entailment under bag semantics, for which so far it was only known that it is in Π_3^P [97]. In [95], Razniewski et al. proposed completeness patterns and defined a pattern algebra to check the completeness of queries. The work incorporated database instances, yet provided only a sound algorithm for completeness check.

[9]Prefix declarations are provided in Appendix A.

We now move on to the Semantic Web. Fürber and Hepp [38] distinguished three types of completeness: ontology completeness, concerning which ontology classes and properties are represented; population completeness, referring to whether all objects of the real-world are represented; and property completeness, measuring the missing values of a specific property. Those three types of completeness together with the interlinking completeness, i.e., the degree to which instances in the dataset are interlinked, are considered to be the bases of the completeness dimension for RDF data sources [114]. Our work considers completeness statements which are built upon BGPs, and hence have more flexibility in expressing completeness (e.g., "complete for all children of the US presidents who were born in Hawaii"). Mendes et al. [73] proposed Sieve, a framework for expressing quality assessment and fusion methods, where completeness is also considered. With Sieve, users can specify how to compute quality scores and express a quality preference specifying which characteristics of data indicate higher quality. Ermilov et al. [36] presented LODStats, a statistics aggregation of RDF datasets published over various data portals such as `data.gov`, `publicdata.eu`, and `datahub.io`. They discussed several use cases that could be facilitated from such an aggregation, including coverage analysis (e.g., most frequent properties and most frequent namespaces of a dataset). As opposed to Sieve and LODStats, our work puts more focus on describing completeness of data sources, and leveraging such completeness descriptions for checking query completeness (and soundness). Galárraga et al. [41] proposed a rule mining system that is able to operate under the Open-World Assumption (OWA) by simulating negative examples using the Partial Completeness Assumption (PCA). The PCA assumes that if the dataset knows some r-attribute of x, then it knows all r-attributes of x. This heuristic was also employed by Dong et al. [35] (called Local Closed-World Assumption in their paper) to develop Knowledge Vault, a Web-scale system for probabilistic knowledge fusion. Our completeness statements, which are based on BGPs, are in fact a generalization of the assumption used in the above work.

3.6. Summary

The availability of an enormous amount of RDF data on the Web calls for better data quality management. Completeness is a crucial quality aspect for RDF data, particularly due to RDF's incomplete nature. In this chapter, we have extended completeness reasoning to be aware with the content of RDF data sources to which completeness statements are given. We have formalized the problem of data-aware completeness entailment and developed a sound and complete algorithm to check the entailment. To increase the practical benefits of our framework, we have identified two fragments of completeness statements: SP-statements, suitable for entity-centric, crowdsourced RDF data sources, and no-value statements, suitable for expressing the non-existence of information in RDF.

In the next chapter, we show how we develop an efficient implementation of completeness reasoning, both in the data-agnostic and data-aware settings.

Chapter 4

Optimizing Completeness Reasoning

Real-world RDF data sources may contain a large amount of data, which is then likely to correspond to a large number of statements needed to describe the completeness of those data sources. Up to this point, we have seen how completeness entailment is formalized and characterized in the data-agnostic setting (see Chapter 2) and data-aware setting (see Chapter 3). Now, the question is how in practice we may perform completeness reasoning, in particular when there are large sets of completeness statements. In this chapter, we develop optimization techniques for the data-agnostic and data-aware completeness reasoning. We also conduct experimental evaluations to show the feasibility of completeness reasoning using our optimizations. The results of data-agnostic reasoning optimizations have been published in [28], whereas those of data-aware reasoning optimizations have been published in [29].

4.1. Optimizing Data-agnostic Reasoning

Here we show how we develop our optimization techniques for data-agnostic completeness reasoning. We first propose the notion of relevant completeness statements wrt. a query, which is potentially useful to reduce the number of completeness statements employed in the reasoning. Then, we describe and evaluate several indexing techniques for the retrieval of relevant completeness statements. Finally, we show, via an experimental evaluation with real query logs from DBpedia, LinkedGeoData, and Semantic Web Dog Food, how the feasibility of data-agnostic completeness reasoning can be improved using the relevance principle.

4.1.1. Relevant Completeness Statements

Before formulating a principle to optimize data-agnostic completeness reasoning, let us first estimate the complexity of the reasoning task. Let $Q = (W, P)$ be a query and $\mathbf{C}$ be a set of completeness statements. According to Theorem 2.10, the task of completeness reasoning is to check whether $T_{\mathbf{C}}(\tilde{P}) = \tilde{P}$, where $T_{\mathbf{C}}$ is the transfer operator wrt. $\mathbf{C}$, 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 $C \in \mathbf{C}$ such that $(s, p, o) \in [\![Q_C]\!]_{\tilde{P}}$ (recall that $T_{\mathbf{C}}(\tilde{P}) = \bigcup_{C \in \mathbf{C}} [\![Q_C]\!]_{\tilde{P}}$). Hence, we only find statements that *potentially* match such a triple (s, p, o).

Let $Q = (W, P)$ be a query, $\mathbf{C}$ be a set of completeness statements, and $maxLn(\mathbf{C})$ be the maximum length (i.e., the maximum number of triple patterns) of statements in $\mathbf{C}$. Take any $C \in \mathbf{C}$; 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 $\mathbf{C}$, leads to the following overall runtime:

$$O(|\mathbf{C}||\tilde{P}|^{maxLn(\mathbf{C})}) \tag{4.1}$$

As customary in the database theory when analyzing the data complexity of query evaluation, we are assuming 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 bounded 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. Thus, we need to find an approach to reduce the number of completeness statements involved in completeness reasoning. According to Theorem 2.10, which characterizes the data-agnostic completeness entailment, for a complete query with n triple patterns, there is a set of no more that n completeness statements that already entails the completeness of that query. Nevertheless, there is no obvious way to identify a priori such a set. Despite this, in the following we establish a principle that allows us 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 "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 introduce 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, $\mathit{const}(C) \subseteq \mathit{const}(Q)$. 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 4.1. *Let C be a completeness statement and $Q = (W, P)$ be a query. If C is not constant-relevant wrt. Q, then $[\![Q_C]\!]_{\tilde{P}} = \emptyset$.*

Proposition 4.1 opens up the problem of how to (efficiently) retrieve constant-relevant statements. In the next subsection, we provide a report of our investigation on retrieval techniques for constant-relevant completeness statements.

4.1.2. Retrieval Techniques for Constant-Relevant Statements

For a set $\mathbf{C}$ of completeness statements, we want to know how to retrieve as efficiently as possible those statements that are constant-relevant wrt. a given query Q. Here, we give an overview of techniques to retrieve such statements.

The statements in $\mathbf{C}$ that are constant-relevant to Q are those all of whose constants appear in Q. We denote this set as $\mathbf{C}_Q$, that is,

$$\mathbf{C}_Q = \{\, C \in \mathbf{C} \mid \mathit{const}(C) \subseteq \mathit{const}(Q) \,\}.$$

To compute $\mathbf{C}_Q$ from $\mathbf{C}$ and Q, is an instance of the well-established *subset querying problem*, which has been investigated by the database and AI communities [51, 54, 102].

The subset querying problem itself is defined as follows: Given a set $\mathbf{S}$ of sets, and a query set S_q, retrieve all sets in $\mathbf{S}$ that are contained in S_q. In our setting, $\mathbf{S}$ consists of the constant sets $\mathit{const}(C)$ of the completeness statements C, while the query set S_q consists of the constants in Q, that is, $S_q = \mathit{const}(Q)$.

We study two retrieval techniques based on specialized index structures for subset querying, namely, inverted indexes and tries. The former is inspired by the approach from the database communities [51], while the latter is from the AI communities [54, 102]. Those approaches were empirically shown to be efficient for their respective subset-querying-based problems. Additionally, we develop a baseline technique using standard hashing. In Subsection 4.1.3, we present experimental evaluations comparing the retrieval time and scalability of the three techniques.

Running Example. Throughout the description below, we will provide examples referring to a set $\mathbf{C} = \{ C_1, C_2, C_3, C_4 \}$ of completeness statements with

- $\mathit{const}(C_1) = \{ a, b \}$,
- $\mathit{const}(C_2) = \{ a, b, c \}$,
- $\mathit{const}(C_3) = \{ a, b, c \}$,
- $\mathit{const}(C_4) = \{ d \}$,

and a query Q with $\mathit{const}(Q) = \{ a, b \}$. It is the case that $\mathbf{C}_Q = \{ C_1 \}$, as C_1 is the only statement in $\mathbf{C}$ all of whose constants are contained in $\mathit{const}(Q)$.

We now describe how these retrieval techniques work and how we implemented them for our experiments. The implementation language was Java. We represent completeness statements using a class `CompletenessStatement`, while constants are simply represented by standard Java strings.

4.1.2.1. Standard Hashing-based Retrieval

In this baseline approach, we 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. We store completeness statements according to their constant sets using a hash map. For each of the $2^{|const(Q)|} - 1$ non-empty subsets of $const(Q)$, we generate a set equality query using the hash map to retrieve the statements with exactly those constants. In our example, the non-empty subsets of $const(Q)$ are $\{a\}$, $\{b\}$, and $\{a, b\}$. Querying for both $\{a\}$ and $\{b\}$ returns the empty set, while querying for $\{a, b\}$ returns the set $\{C_1\}$. Taking the union of these three results gives us $\{C_1\}$ as the final result.

Implementation. To index the statements, we use a standard Java `HashMap`. To each statement, we associate a key that uniquely represents the set of its constants. We do that by creating a lexicographically ordered sequence of the constants in the statement. We use the standard Java `List` to represent sequences and the `sort` method of the Java `Collections` class for sorting. Then, for such a key, the value in the hash map is the set of all statements having exactly the constants mentioned in the key. To compute $\mathbf{C}_Q$, we generate all sequences corresponding to the nonempty subsets of $const(Q)$, retrieve the values to which they are mapped using the `get` method of the `HashMap`, and take the union of the values.

4.1.2.2. Inverted Indexing-based Retrieval

Inverted indexes have been originally developed by the information retrieval community for search engine applications [116]. In the information retrieval domain, an inverted index is a data structure that maps a word to the set of documents containing that word. Inverted indexes are typically used for finding documents containing all words in a search query, that is, for superset querying.

In database applications, inverted indexes are also used for subset querying. In object-oriented databases, objects may have set-valued attributes. Given an attribute and a query set, one may want to find all the objects whose set of attribute values is contained in the query set. Helmer and Moerkotte [51] compared indexing techniques for an efficient evaluation of set operation queries (i.e., subset, superset and set equality) involving low-cardinality set-valued attributes. The indexing techniques they considered were inverted indexes and three other techniques that are signature-based (i.e., sequential signature files, signature trees, and extendible signature hashing). There, an inverted index maps each value to the objects whose set-valued attribute contains that value. Their experimental evaluations showed that in terms of retrieval costs, inverted indexes overall performed best.

Formalization. Now we show how we develop our retrieval technique based on inverted indexes, adapted from [51]. For a set $\mathbf{C}$ of completeness statements, we let $\mathbf{P} = \bigcup_{C \in \mathbf{C}} \mathit{const}(C)$ be the set of all constants in $\mathbf{C}$. We define the map $M \colon \mathbf{P} \to 2^{\mathbf{C}}$ such that $M(p) = \{ C \in \mathbf{C} \mid p \in \mathit{const}(C) \}$ for every constant $p \in \mathbf{P}$. In other words, M maps each constant occurring in $\mathbf{C}$ to the set of completeness statements in $\mathbf{C}$ containing that constant. We call such a map an *inverted index*. The inverted index M of our example is shown below.

Constants	Completeness Statements
a	C_1, C_2, C_3
b	C_1, C_2, C_3
c	C_2, C_3
d	C_4

We now want to retrieve constant-relevant statements using inverted indexes. As a first attempt, for a query Q and the inverted index M of a set $\mathbf{C}$ of completeness statements, we consider the set union $\bigcup_{p \in \mathit{const}(Q)} M(p)$ of the mappings of the constants occurring in the query. In our example, this is the set $\{ C_1, C_2, C_3 \}$. However, though the resulting set is smaller than the original set $\mathbf{C}$, it is still bigger than $\mathbf{C}_q$, since it contains statements that are not constant-relevant (i.e., C_2 and C_3).

Now, instead of the set union, let us consider bag union. For a start, assume that $M(p)$ is now a bag that contains as many copies of a statement C as there are occurrences of the constant p in C. In our running example, each $M(p)$ still contains at most one copy of a statement. Next, we take

$$B_Q = \biguplus_{p \in \mathit{const}(Q)} M(p),$$

which is the bag of all statements that have at least one constant in Q, and where a statement occurs as many times as it has occurrences of constants appearing in the query Q. With respect to our example, $B_Q = M(a) \uplus M(b) = \{\!| C_1, C_1, C_2, C_2, C_3, C_3 |\!\}$. Let us analyze which statements are constant-relevant. The statement C_1 occurs twice in B_Q and has length 2, hence, all its constants appear in the query Q. However, the statements C_2 and C_3 both have length 3, but occur only twice in B_Q. This means that they have other constants that do not appear in the query Q and thus, they are not constant-relevant. Therefore, we conclude that $\mathbf{C}_Q = \{ C_1 \}$.

We can generalize our example to arrive at a characterization of the set $\mathbf{C}_Q$. The example shows that we need to count the occurrences of completeness statements in B_Q. We denote the count of a statement C in B_Q by $\#_C(B_Q)$. As seen from the example, those statements whose number of occurrences is the same as the number of constants are the constant-relevant ones. In this case, for a statement C, we take the bag version of $\mathit{const}(C)$. Then, $\mathbf{C}_Q$ satisfies the equation

$$\mathbf{C}_Q = \{ C \in B_Q \mid \#_C(B_Q) = |\mathit{const}(C)| \}.$$

Implementation. We observe from the formalization that the crucial operations for the retrieval technique using inverted indexes are bag union and count. We chose the Google Guava library[1] as it provides a bag implementation in Java with the class `HashMultiset`, which includes as methods the bag union and count. To implement the inverted index, we use the Java `HashMap`. The index maps each constant p to the `HashMultiset` representing the bag of completeness statements containing that constant (i.e., $M(p)$). As shown in the formalization above, to retrieve B_Q, we perform a bag union, using the `addAll` method of the `HashMultiset`, of the map values of the constants in Q. Then, to retrieve the set $\mathbf{C}_Q$ of constant-relevant statements, we count the number of occurrences of the statements in B_Q using the `count` method of the `HashMultiset` and check if the count is the same as the size of the statement.

4.1.2.3. Trie-based Retrieval

A trie, or a prefix tree, is an ordered tree for storing sequences, whose nodes are shared between sequences with common prefixes. Tries have been adopted for set-containment queries in the AI community by Hoffmann and Koehler [54] and Savnik [102]. Both studies showed

[1] `https://github.com/google/guava`

by means of empirical evaluations that tries can be used to efficiently index sets, and perform subset and superset queries upon those sets. Set operations are essential in AI applications, including the matching of a large number of production rules and the identification of inconsistent subgoals during planning.

Formalization. We show how to adapt tries as in [54, 102] to our setting. The sequences we consider are sequences of constants that are ordered lexicographically. For a set $\mathbf{C}$ of statements, we define $\mathbf{S_C}$ as the set containing for each statement in $\mathbf{C}$ the corresponding sequence of constants. The *trie* $\mathbf{T_C}$ over the set $\mathbf{S_C}$ of sequences is the tree whose nodes are the prefixes of $\mathbf{S_C}$, denoted as $\mathit{Pref}(\mathbf{S_C})$, where each node $\bar{s} \in \mathit{Pref}(\mathbf{S_C})$ has a child $\bar{s} \cdot p$ iff $\bar{s} \cdot p \in \mathit{Pref}(\mathbf{S_C})$, where p is a constant. On top of this trie, we define $M \colon \mathit{Pref}(\mathbf{S_C}) \to 2^{\mathbf{C}}$ as the mapping that maps each prefix to the set of statements whose constants are exactly those in the prefix.

In our example, we have that $\mathbf{S_C} = \{ (a,b), (a,b,c), (d) \}$ and $M = \{ (a,b) \mapsto \{ C_1 \}, (a,b,c) \mapsto \{ C_2, C_3 \}, (d) \mapsto \{ C_4 \} \}$. For simplicity, we left out mappings with the empty value in M. A graphical representation of the trie $\mathbf{T_C}$ is shown below, which also shows the map value of each node wrt. M.

$$\begin{tikzcd}
 & () \arrow[dl, no head] \arrow[dr, no head] & \\
(\mathrm{a}) \arrow[d, no head] & & (\mathrm{d}) : \{ C_4 \} \\
(\mathrm{a},\, \mathrm{b}) : \{ C_1 \} \arrow[d, no head] & & \\
(\mathrm{a},\, \mathrm{b},\, \mathrm{c}) : \{ C_2, C_3 \} & &
\end{tikzcd}$$

Having built a trie from completeness statements, we now want to retrieve the constant-relevant statements wrt. a query. Let us do that for our example. Consider the trie $\mathbf{T_C}$ as above. As $\mathit{const}(Q) = \{ a, b \}$, the sequence of $\mathit{const}(Q)$ is therefore $\bar{s}_Q = (a, b)$. The key idea behind our retrieval is that we visit nodes that are subsequences of the query sequence and collect the map values of the visited nodes wrt. M. We start at the root of $\mathbf{T_C}$ with the query sequence (a, b) and an empty set of constant-relevant statements. The root node is trivially a subsequence of $\bar{s}_Q$ and the mapping of the root obviously returns the empty set. Thus, our set of constant-relevant statements is still empty.

At this position, we have two options. The first is to retrieve from $\mathbf{T_C}$ all the subsequences containing the head of the current query sequence, that is, the constant a. By the trie structure, all such subsequences reside in the subtree of $\mathbf{T_C}$ rooted at the concatenation of the root of the current trie and the head of the current query sequence. We then proceed down that subtree. To proceed down, the head of the query sequence has to be removed. Therefore, our current query sequence is now (b). As the map value of the root (a) of the current trie is empty, we still have an empty set of constant-relevant statements. From this position, we try to visit the subsequences in $\mathbf{T_C}$ that not only contain a, but also one additional constant from the current query sequence. Therefore, we continue proceeding down the subtree rooted at (a, b), which is the concatenation of the root of the current trie and the head of the current query sequence. From the mapping result of the root (a, b), the set of constant-relevant statements is now $\{ C_1 \}$. Since our current query sequence is now the empty sequence, we do not proceed further.

Now, let us pursue the second option. We stay at the position at the root of $\mathbf{T_C}$, while simplifying $\bar{s}_Q$ by removing the head of the query sequence, making it now (b). In this case, we want to visit all the subsequences in the trie $\mathbf{T_C}$ that do not contain the constant a, if they exist. Now, we try to proceed down the subtree rooted at the concatenation of the root of the current trie and the head of the current query sequence. This means we have to proceed down the subtree rooted at (b). Since it does not exist, we stay with the current trie and remove again the head of the query sequence. As the query sequence is now the empty sequence, we do not go further and finish our whole tree traversal. As a final result, we have our set of constant-relevant statements which contains only C_1.

From our example, we now formalize the retrieval of constant-relevant statements using tries. We can decompose a non-empty sequence $\bar{s} = (p_1, \ldots, p_n)$ into the head p_1 and the tail $(p_2, \ldots, p_n)$. For a sequence $\bar{s}$ and a trie $\mathbf{T}$, we define $\mathbf{T}/\bar{s}$ as the subtree in $\mathbf{T}$ rooted at the node $\bar{s}$. Note that $\mathbf{T}/\bar{s}$ is the empty tree $\perp$ if such a subtree does not exist. We define $cov(\bar{s}_Q, \mathbf{T_C})$ as the set of completeness statements in $\mathbf{C}$ whose sequences of their constants are subsequences of $\bar{s}_Q$, which are not necessarily contiguous. It follows from this definition that $cov(\bar{s}_Q, \mathbf{T_C}) = \mathbf{C}_Q$. Given a subsequence $\bar{s} = p \cdot \bar{s}'$ of $\bar{s}_Q$ and a subtree $\mathbf{T}$ of $\mathbf{T_C}$, we observe that the function *cov* satisfies the following recurrence property:

$$cov(\bar{s}, \mathbf{T}) = \begin{cases} \emptyset & \text{if } \mathbf{T} = \bot \\ M(root(\mathbf{T})) & \text{if } \bar{s} = () \\ M(root(\mathbf{T})) \cup cov(\bar{s}', \mathbf{T}/(root(\mathbf{T}) \cdot p)) \cup & \text{otherwise.} \\ \qquad cov(\bar{s}', \mathbf{T}) & \end{cases}$$

The recurrence property has two base cases: when the trie is empty, then simply the empty set is returned; and when there is no element left in the sequence $\bar{s}$ (i.e., the trie traversal stops), the *cov* function returns the set of completeness statements associated with the sequence $root(\mathbf{T})$. Now for the recursive case, there are three components involved. The first one is simply returning the set of completeness statements associated with $root(\mathbf{T})$. The second and third ones correspond to how the trie is traversed: both make the *cov* calls with the tail $\bar{s}'$ of $\bar{s}$ as the call's sequence, but the second case is over the subtree $\mathbf{T}/(root(\mathbf{T}) \cdot p)$ while the third one is over the same trie $\mathbf{T}$.

Note that in the above property, as also observed in [54], the function *cov* performs pruning: when a subtree in $cov(\bar{s}, \mathbf{T}/(root(\mathbf{T}) \cdot p))$ does not exist, we cut out all the recursion call possibilities if the subtree existed. Let us give an illustration. For a query sequence $\bar{s}_Q = (p_1, \ldots, p_n)$ of length n, there are at most 2^n possible subsequences. However, half of them (those containing p_1) lie in the tree rooted at the node (p_1). If there is no node (p_1), the size of the search space is immediately reduced to 2^{n-1}.

Implementation. We represent sequences of constants in $Pref(\mathbf{S_C})$ using the Java `List<String>` class. For implementing the trie $\mathbf{T_C}$, we create a class `Trie`. For the trie nodes, we create `TrieNode` objects labeled with sequences of constants. A `TrieNode` has a hash map that maps the sequences of constants of the `TrieNode`'s children to the corresponding `TrieNode` objects. Initially, a `Trie` has a `TrieNode` object as its root with an empty sequence as the label. For every insertion of a sequence of the constants of a completeness statement, we recursively generate children of `TrieNode` objects starting from the root to the leaf node with that sequence as the label. This generates a path of `TrieNode` objects labeled with the prefixes of that sequence. `TrieNode` objects are shared between sequences with the same prefixes. To implement the map M for the trie, a Java `HashMap` similar to the one in the implementation of the standard hashing technique is created.

For the retrieval, we implemented a recursive method based on the recurrence property of the *cov* function. In the method, for each visited node, we use the `HashMap` of M to map the label of the node to its corresponding set of completeness statements. All the mapping results are collected in a standard Java set which at the end of the method call will be our set $\mathbf{C}_Q$ of constant-relevant statements.

4.1.3. Experimental Evaluation of the Retrieval Techniques

We have discussed the constant-relevance principle as a means to prune the set of completeness statements. We have also introduced three retrieval techniques of constant-relevant statements, based on standard hashmaps, inverted indexes, and tries as the underlying index structures. We now report on experiments that comparatively evaluated those three techniques. More specifically, the experiments aim to analyze: the runtime and scalability of the retrieval techniques according to various parameters that contribute to the overall runtime of completeness reasoning, as analyzed in Eq. (4.1) (i.e., number of completeness statements, length of completeness statements, and length of queries); and the cost of completeness reasoning without vs. with the optimization technique.

4.1.3.1. Experimental Setup

We created a framework for the experiments consisting of two components: a completeness reasoner and a generator of statements and queries. We implemented the framework in Java using the Jena library.[2]

The completeness reasoner includes implementations of the three retrieval techniques as described before and supports reasoning optimizations based on the constant-relevance (that is, instead of considering all statements in $\mathbf{C}$, the optimized technique considers only the statements in $\mathbf{C}_Q$).

[2] `http://jena.apache.org/`

To gain flexibility in setting the experiment parameters, we randomly generate queries and sets of completeness statements. In our experiment, we would expect that the bigger a data source, the more completeness statements are declared over that source. We want to consider also the sensitivity of each retrieval technique to the length of the completeness statements and the query. Thus, we choose the following experiment parameters:

- number of completeness statements (N_c),
- maximum length of completeness statements (L_c), and
- length of queries (L_q).

To evaluate the retrieval techniques, we want to observe the influence of each parameter on the retrieval time. Thus, we set up three scenarios, where in each we keep two of the parameters fixed and vary the remaining one. As our reference for setting the default values for the parameters, we take DBpedia [10], one of the most popular and largest RDF data sources, as an approximation of the realistic parameter values. From English Wikipedia, DBpedia extracted around 580 million RDF triples.[3] If we assume that $\frac{1}{5}$ of the triples are captured by completeness statements, and that each statement covers 100 triples, then DBpedia would have 1,160,000 completeness statements. Therefore, we set the default value $N_c =$ 1,000,000. The length of queries is chosen based on the statistics of SPARQL queries over DBpedia. Arias et al. [8] found that 97% of DBpedia queries are of length less than or equal to 3. Therefore, we choose 3 as the default length for short queries. On the other hand, 99.9% of queries over DBpedia had length less than or equal to 6, so a length of 6 stands for relatively long queries. So, there are two default values for query length: $L_q = 3$ for the short ones, and $L_q = 6$ for the long ones. As for the default value of L_c, we set it to 6, to have a variation of completeness statement length from 1 to 6, which covers the query length.

The experiments were run on a standard laptop under Windows 8 with Intel Core i5 2.5 GHz processor and 8 GB RAM. For each combination of parameter values, we ran the experiment 20 times to obtain reliable results (i.e., low variance if we performed the experiments again), and took the median of the runtimes.

[3] `http://lists.w3.org/Archives/Public/public-lod/2014Sep/0028.html`

Random Generation of Statements and Queries. The statements and queries for the experiments have been generated randomly with a uniform distribution of the IRIs for constants. Again, we take DBpedia as our reference. DBpedia has about 2,700 properties and 4.5 million entities, We approximate the number of constant IRIs in the predicate position from the number of properties of DBpedia, that is, 2,500, and the number of constant IRIs in the subject or object position from about $\frac{1}{5}$ of the number of DBpedia entities, that is, 1,000,000. The generated statements were of the form $Compl(P)$, while the generated queries were of the form $(var(P), P)$, that is, all variables in the body were distinguished. Generating the statements and queries is essentially generating triple patterns, which serve as their building blocks.

The triple patterns of a statement are generated as follows. First, we pick a random length between 1 and L_c. Then we randomly choose the predicates of the triple patterns, where repetitions are allowed. Next, for this collection of predicates, we generate fully-formed triple patterns. To do that, we instantiate the subjects and objects of triple patterns, by constants or variables. For the instantiation by constants, we randomly take IRIs, and the constants can be reused across triple patterns. We do not limit the possibility to introduce new variables, but again variables can be reused among triple patterns. We generate variables in such a way that there is no cross-product join between triple patterns of the statement, that is, the triple patterns with variables form one connected component. Together, the generated triple patterns become the pattern P for that statement. We repeat this process until there are N_c randomly generated statements. We generate triple patterns for the query of length L_q in the similar way.

4.1.3.2. Results and Discussion

We now show the experimental results comparing the retrieval time of the three techniques. In each scenario, we vary one of these parameters: number of statements, maximum length of completeness statements, and query length. Moreover, we also compare the runtime of completeness reasoning with vs. without the constant-relevance principle.

Influence of the Number of Completeness Statements. In this scenario, we vary the parameter N_c within the range of 100,000 – 1,000,000. Figure 4.1 shows the resulting retrieval times. The left figure is for short queries and the right figure for long ones. The y-axis is in log-scale. As can be clearly seen, inverted indexing is generally slower and less scalable than the other techniques. It is on average 53× slower than tries for short queries and 3× slower than standard hashing for long queries. The performance comparison of standard hashing and tries, however, depends on the length of the queries. For short queries, standard hashing is slightly faster. For long queries, the tries technique is faster.

One possible reason why inverted indexing is slow is that at an intermediate step it has to process all statements whose constants overlap with the constants of the query. Hence, with inverted indexing the probability for a completeness statement to be processed in the retrieval is much larger than for other retrieval techniques. The other techniques only process statements whose constants are clearly contained in the query constants. For long queries, the tries perform better than the standard hashing. This is likely due to the subsequence pruning of tries as described in Subsection 4.1.2.

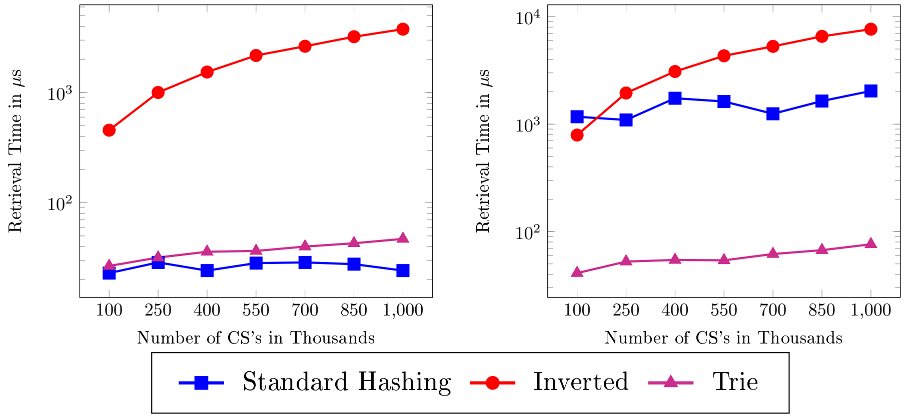


Figure 4.1. Increasing the number of completeness statements for short (left) and long queries (right)

Influence of the Length of Completeness Statements. In this scenario, we vary the maximum length L_c of completeness statements from 1 to 6. Figure 4.2 shows the resulting retrieval times. Interestingly, the retrieval time for inverted indexing increases, while the time for tries even decreases. Basically, the retrieval time for standard hashing remains constant, though showing a little oscillation with no clear pattern. We notice that for short queries, standard hashing performs best, whereas for long queries, tries perform best. Again, inverted indexing performs the worst among all the retrieval techniques.

These graphs demonstrate the fundamental difference between the inverted indexes and the tries. In the inverted indexes, a completeness statement with just a single constant overlapping with the query is included in the bag union, to be checked if the statement's occurrences in the union are the same as its length. Thus, the longer the completeness statement, the more probable it is for the statement to be included in the bag union. This does not happen with the trie-based technique as it only processes statements all of whose constants are contained in the query. When a statement becomes longer, the probability of the statement to be processed by the tries technique decreases. That the growth is nearly constant for standard hashing, is likely due to evaluating always the same set equality queries.

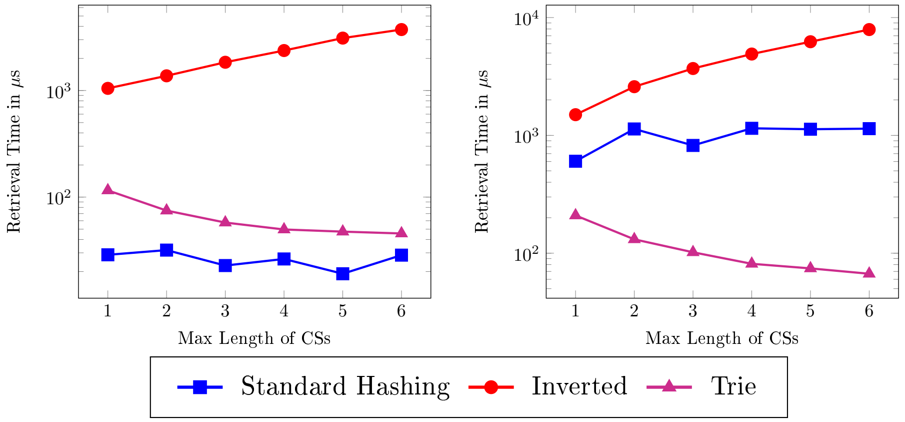


Figure 4.2. Increasing maximum length of completeness statements for short (left) and long queries (right)

Influence of the Query Length. In this scenario, we vary the query length L_q from 1 to 6. Figure 4.3 shows the results of this experiment. From the graph, we can see that for all techniques, the retrieval time increases with the query length, though at different rates. For standard hashing, it grows exponentially, whereas for the other techniques, it only grows linearly.[4] In the beginning, the standard hashing

[4]Note that the graph is displayed in log-scale on y-axis.

technique performs better than the tries one. However, from $L_q = 4$ the standard hashing technique starts to perform worse. At $L_q = 6$, standard hashing is about 14× slower than tries. We observe a similarity between the asymptotic growth of inverted indexing and tries, though on an absolute scale the tries technique clearly performs better.

As expected, standard hashing does not perform well for long queries due to its exponentially many set equality queries. The tries technique, though potentially having exponential growth in the worst case, performs better than standard hashing. This is most likely due to its pruning ability over subsequences of query constants.

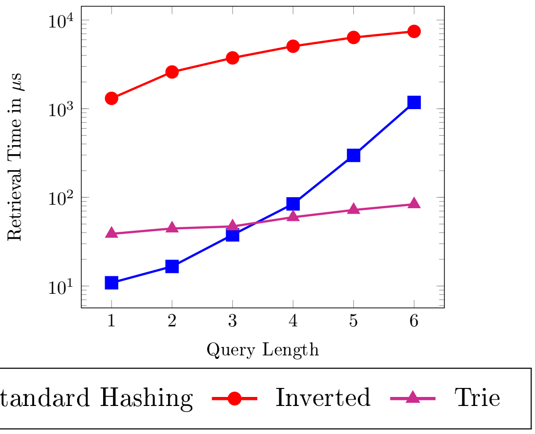


Figure 4.3. Increasing the query length

Reasoning with the Constant-Relevant Filtering. This scenario differs from the above in that now we compare the cost of completeness reasoning without and with the optimization technique. We show that applying the constant-relevance principle can considerably reduce the overhead incurred by completeness reasoning.

To measure this overhead, we perform experiments that compare the runtimes of plain completeness reasoning and of reasoning based on constant-relevance. For the reasoning based on constant-relevance, we use the standard hashing retrieval technique as it shows relatively good performance in our previous experiments. All the parameter values are the default ones: $N_c = 1{,}000{,}000$, and $L_c = 6$, while we still distinguish between short queries ($L_q = 3$) and long queries ($L_q = 6$). In the experiments we measure the reasoning time for plain completeness reasoning and the reasoning plus the retrieval time for the completeness reasoning based on constant-relevance.

Table 4.1. Comparison of the runtime median for plain completeness reasoning and constant-relevance based (optimized) reasoning

Query Types	Plain Reasoning	Optimized Reasoning
Short	145,773 ms	1.3 ms
Long	146,095 ms	4.1 ms

Now we discuss the experimental results. Table 4.1 lists the median of runtimes of plain completeness reasoning and constant-relevance based completeness reasoning. We note that completeness reasoning based on constant-relevance is considerably faster than the plain one (i.e., milliseconds vs. minutes, respectively).

Completeness reasoning with the constant-relevance principle is fast, with runtimes between 110,000 times (for short queries) and 35,000 times (for long queries) faster than that without constant-relevance. This is due to the fact that much fewer completeness statements are considered for the reasoning using the constant-relevance principle. For short queries, there are on average about 49 constant-relevant completeness statements, whereas for long queries, there are on average about 105 constant-relevant statements. On the other hand, the original set contains 1 million completeness statements.

Conclusions of the Experiments. From the experiments we conclude that for short queries, our baseline approach, the standard hashing, shows the best performance despite its simplicity. However, for long queries, the tries technique performs better. The baseline approach suffers from its exponential blow up for long queries. The inverted indexes are not suitable for the retrieval task for both short and long queries. Moreover, on an absolute scale, the retrieval time of the retrieval techniques only takes up to about a few milliseconds. This shows that the retrieval process does not add a significant overhead to completeness reasoning. Also, we have seen that using the constant-relevance principle can considerably speed up completeness reasoning, as demonstrated in Table 4.1.

4.1.4. Experimental Evaluation of Data-agnostic Reasoning

In this subsection, we aim to investigate the performance of completeness reasoning for realistic cases based on several RDF data sources: DBpedia (DBP), Semantic Web Dog Food (SWDF), and Linked Geo Data (LGD). Our investigation in finding a retrieval technique for constant-relevant completeness statements showed that despite its simplicity, standard hashing can outperform the inverted indexing [51] and the tries [54, 102] technique for queries with the length of up to 3, accounting for 97% of real-world queries on DBpedia [8], with the worst case of retrieval time of only 2 ms. Thus, we concentrate our analysis on standard hashing. We can break down the process of completeness reasoning into two main components: (*i*) the hashmap lookup to retrieve constant-relevant statements; and (*ii*) the $T_{\mathbf{C}}$-application of all constant-relevant statements over the prototypical graph $\tilde{P}$ as per Theorem 2.10. 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 components, the hashmap lookup and the $T_{\mathbf{C}}$-application, influence the overall completeness reasoning time?

Experimental Setup. We created a framework for the experiments in Java using the Apache Jena library, an open source Semantic Web library.[5] 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`. The two ingredients that characterize our setting were queries and completeness statements.

As for the queries, we used openly available real query logs of RDF data sources across various domains, i.e., DBpedia, Semantic Web Dog Food, and LinkedGeoData, provided in the Linked SPARQL Queries (LSQ) dataset [100]. We extracted `SELECT` queries in the conjunctive fragment, which account for about 40% of the total number of `SELECT` queries, giving us around 465,000 queries in total.[6]

As for the completeness statements, for each query we took its full BGP P and constructed a completeness statement $Compl(P)$. Via

[5]`http://jena.apache.org/`
[6]As of May 22, 2016

query homomorphism techniques [19], we removed redundant completeness statements, i.e., completeness statements whose `CONSTRUCT` query representations are equivalent. In total, there were about 400,000 completeness statements generated. Observe that by construction, all queries are guaranteed to be complete. The experiment framework (incl. the source code) is available online at `http://completeness.inf.unibz.it/completeness-experiment/`.

We distinguished between three cases of the experiments, depending on the endpoint of the queries: DBP, SWDF, or LGD. We measured completeness reasoning time of the queries of each case. The experiments were run on a standard laptop under Windows 8.1 with Intel Core i5-2435M 2.4 GHz processor and 8 GB RAM. Furthermore, for each query we also took the query evaluation time, which is already provided by the LSQ dataset. The experiment machine for query evaluation was with 16 GB RAM and a 6-Core i7 3.40 GHz CPU running Ubuntu 14.04.2 using Virtuoso 7.1 [100]. Note that the machine for query evaluation was relatively better than our machine for completeness reasoning.

Results and Discussion. Table 4.2 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 most queries are short, close to 1 triple pattern, with a slight exception of LGD queries whose average length is in the middle between 1 and 2 triple patterns. The average of completeness reasoning time for all cases is always below 0.2 ms.

Table 4.2. Overview of the experiment results, where N_Q is the number of queries, $N_\mathbf{C}$ 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, and t_{QE} is the average query evaluation time

Endpoint	N_Q	$N_\mathbf{C}$	$\lvert Q\rvert$	t_{CR}	t_{QE}
DBP	334,304	331,294	1.13	0.086 ms	18.8 ms
LGD	108,611	44,505	1.54	0.127 ms	36.2 ms
SWDF	22,590	21,616	1.22	0.056 ms	8.3 ms

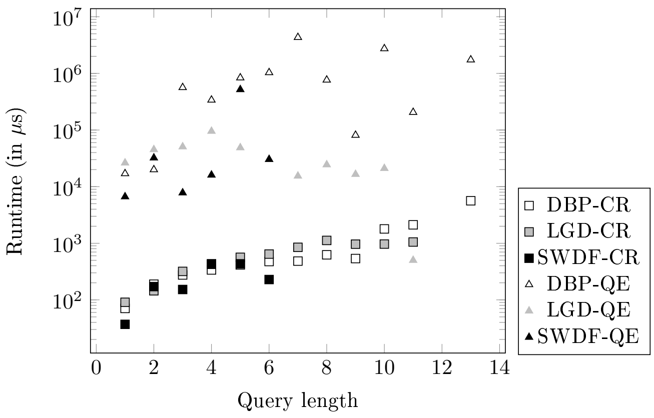


Figure 4.4. Comparison of query length to completeness reasoning (CR) time and query evaluation (QE) time

To get an idea on the performance comparison with plain completeness reasoning (where *all* completeness statements are considered in reasoning), we took the first query for each case and performed completeness reasoning, measuring 4,600 ms, 700 ms, and 600 ms for DBP, LGD, and SWDF, respectively.[7] Thus, we have a considerable speed-up by using the constant-relevance principle, up to 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 respect to query evaluation, completeness reasoning overall only adds a little overhead to query evaluation time, that is, 0.5% on average.

Figure 4.4 shows how the 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 all cases, clearly query evaluation takes much longer than completeness reasoning by several orders of magnitude. Note that in all three query logs we used, most queries have short length, for instance, there are only fewer than 10 queries for each group of DBpedia queries with length greater than 6. Also, the worst case of completeness reasoning

[7]Note that for the other queries, all statements also have to be considered, hence reasoning time would not be much different.

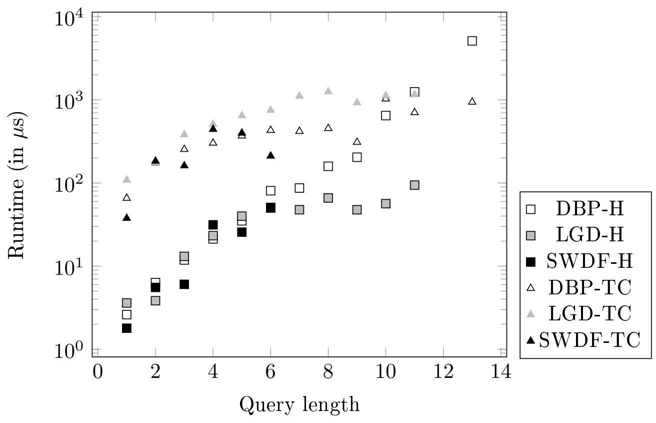


Figure 4.5. Distribution of hashmap lookup time (H) and $T_{\mathbf{C}}$-application time (TC) in completeness reasoning across different query length

time in the figure is only 5.6 ms (where the query length equals 13), which we consider very reasonable.

Regarding the runtime for completeness reasoning with the constant-relevance principle, we can break this up into the time needed for the hashmap lookup for constant-relevant statements, and the time for the $T_{\mathbf{C}}$-application of those constant-relevant statements. Figure 4.5 shows how they distribute. As seen from the figure, the growth of the hashmap lookup time is exponential in the query length, whereas the growth of the $T_{\mathbf{C}}$-application time appears to be roughly linear. We also observe that while initially $T_{\mathbf{C}}$-application takes longer, when queries become longer, the hashmap lookup time starts to dominate the completeness reasoning time, that is, starting from queries of length 11 for the DBpedia case (but such queries are a few). This means that from short- to medium-length queries, the fact that the time for the lookup is exponential is of little importance.

Finally, Figure 4.6 (with linear scale on y-axis) provides an idea on how query length relates with the number of constants in queries and the number of constant-relevant statements, respectively. In the upper figure, it can be seen that the number of constants grows linearly with respect to query length with a few exceptions. This is likely to be the reason for the exponential growth of hashmap lookup time in Figure 4.5, since the hashmap lookup depends exponentially on the number of constants. From the lower figure, we can infer that the query logs contain relatively many similar (sub)-queries since there

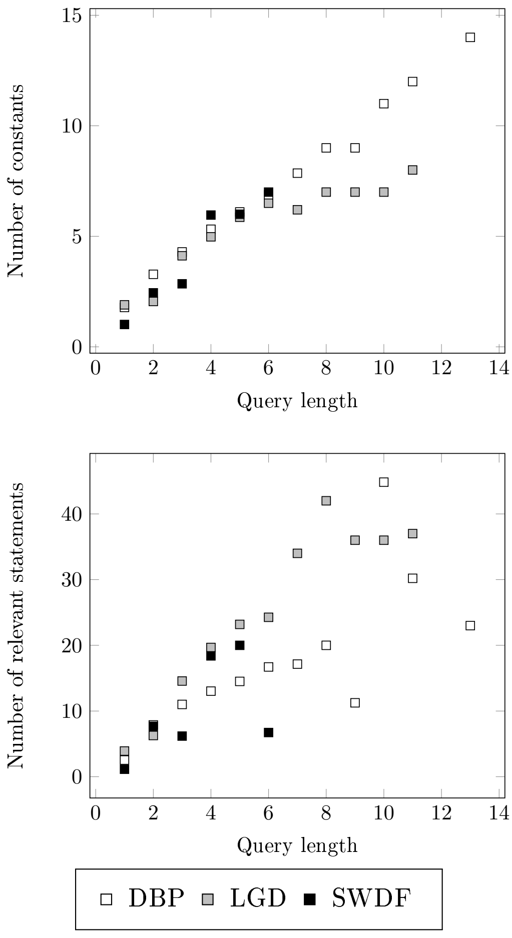


Figure 4.6. Comparison of query length to number of constants (upper) and number of relevant statements (lower)

are quite a number 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. For queries up to length 6, there are at most 25 relevant statements, and this number does not grow much, as the maximum number is 45. There is a weak positive correlation between the query length and the number of relevant statements. By and large, the trend of this figure matches the trend of the $T_{\mathbf{C}}$-application time in Figure 4.5, due to the linear relationship between the number of relevant statements and the $T_{\mathbf{C}}$-application time.

Conclusions of the experiments. We have evaluated completeness reasoning in practical settings based on real query logs from DBpedia, SWDF, and Linked Geo Data SPARQL endpoints. We observed that completeness reasoning with the constant-relevance principle can be done quickly, with the worst case of 5.6 ms. Compared with query evaluation time, completeness reasoning only adds a little overhead, just about 0.5% on average. Also, the performance of completeness reasoning tends to be positively correlated with query length. Furthermore, for short- to medium-length queries, the $T_{\mathbf{C}}$-application time, which grows linearly, dominates the completeness reasoning time, whereas for long queries, the hashmap lookup time, which grows exponentially, dominates the reasoning time. Hypothetically, a possible weakness of this constant-relevance approach might occur when there are a large number of constants in a query (e.g., 32 constants) due to the exponential blowup of the set-equality queries generated. From the query logs, however, long queries are rare and also, queries have at most 14 constants, which are still manageable.

4.2. Optimizing Data-aware Reasoning

For the data-aware setting, reasoning needs access to the data graph. The previous approach to optimization of data-agnostic reasoning, which leaves out statements whose terms are not among the terms of the query, is no more applicable, since parts of the statements can now be mapped to the data graph. We present a new algorithm, which improves upon an earlier one for completeness checking in Chapter 3.

4.2.1. Completeness Templates and Partial Matching

In this subsection, we introduce completeness templates, template-based transfer operator, and partial matching as techniques to optimize data-aware completeness reasoning. Completeness templates are inspired by natural language completeness statements available on the Web, which are usually about similar topics. Then, by exploiting that a template represents many statements, we can leverage query evaluation for simultaneous processing of statements. Finally, partial matching is crucial for filtering out irrelevant templates wrt. the query we want to check for completeness.

Completeness Templates. We represent similar completeness statements by so-called *completeness templates*. Such templates support users in creating completeness statements of similar topics, as they occur for instance in IMDb, which reports completeness for movie cast and crew[8] or in OpenStreetMap, which uses a wiki to record the completeness of objects in different areas.[9] A completeness template is a 3-tuple $\tau = (C, V_\tau, \Omega)$, where C is a completeness statement, $V_\tau \subseteq \mathit{var}(C)$ is a set of variables, called *meta-variables*, and Ω is a set of mappings from V_τ to terms (i.e., IRIs or literals). We also refer to the BGP of the completeness statement C of the template τ as P_τ. As an example of a completeness template, we generalize the statement set

$$\{\, \mathit{Compl}((\mathit{ger}, \mathit{lang}, ?l)), \ldots, \mathit{Compl}((\mathit{spa}, \mathit{lang}, ?l)) \,\},$$

to the template $(\mathit{Compl}((?c, \mathit{lang}, ?l)), \{?c\}, \Omega)$, where $\Omega = \{\, \{\, ?c \mapsto \mathit{ger} \,\}, \ldots, \{\, ?c \mapsto \mathit{spa} \,\} \,\}$. A template $\tau = (C, V_\tau, \Omega)$ represents the statement set $\mathbf{C}_\tau = \{\, \mathit{Compl}(\mu P_C) \mid \mu \in \Omega \,\}$, obtained by instantiating C with the mappings in Ω. This definition naturally extends to sets of completeness templates. Note that a completeness statement C can be expressed as the completeness template $(C, \emptyset, \{\mu_\emptyset\})$ where $\mu_\emptyset$ is the mapping with the empty domain.

Template-based Transfer Operator. A key part of the algorithm for checking completeness, given a statement set $\mathbf{C}$ and a data graph G, is to identify the crucial part P_0 of P, that is, the maximal subset $P_0 \subseteq P$ such that $\tilde{P}_0 \subseteq T_{\mathbf{C}}(\tilde{P} \cup G)$. Given a set $\mathcal{T}$ of completeness

[8] See e.g., `http://www.imdb.com/title/tt0105236/fullcredits`

[9] See e.g., `http://wiki.openstreetmap.org/wiki/Abingdon`

templates, analogously to Eq. (3.1), such a part satisfies the equation

$$P_0 = P \cap \tilde{id}^{-1}(T_{\mathbf{C}_\mathcal{T}}(\tilde{P} \cup G)). \tag{4.2}$$

A baseline approach to compute P_0 in Eq. (4.2) is to instantiate templates to yield completeness statements, and then apply the $T_{\mathbf{C}}$-operator wrt. the statements. This may be costly if there are many instances of those templates. Now, templates allow us to leverage query evaluation for data-aware completeness reasoning by exploiting that a template represents many statements. Essentially, to check whether the $T_{\mathbf{C}}$-operator maps a triple in $\tilde{P}$ by an instantiation of a template τ, we first evaluate P_τ (by treating the meta-variables like variables) over the union graph $\tilde{P} \cup G$, with the condition that at least one triple pattern in P_τ is mapped to a triple in $\tilde{P}$ (since otherwise the mapping does not contribute to P_0), and verify in a second step which of the resulting mappings are compatible with the instantiations of the template τ. In this way, all instances of τ can be processed simultaneously.

To formalize the above idea, we first define prioritized evaluation of a BGP over a pair of graphs (G_1, G_2). In such an evaluation, we consider the first graph G_1 as the *mandatory* and the second as the *optional* graph, which means that at least one triple pattern of the BGP is mapped to a triple of G_1, while there is no need to map any triple pattern to G_2. Formally, *prioritized evaluation* of a BGP P over (G_1, G_2) is defined as $[\![P]\!]_{(G_1,G_2)} = \{\, \mu \mid \mu \in [\![P]\!]_{G_1 \cup G_2}$ and $\mu P' \subseteq G_1$ for some $P' \subseteq P, P' \neq \emptyset \,\}$. So, in our case of completeness checking, the mandatory graph will be the frozen BGP $\tilde{P}$ and the optional graph will be the data graph G.

Example 4.2. Consider the BGP $P_{usa} = \{(usa, lang, ?l)\}$, the graph

$$\begin{aligned} G_{org} = \{&(org_1, founder, ger), (ger, lang, de),\\ &(org_2, founder, usa), (org_2, founder, ger)\},\end{aligned}$$

and the completeness template $\tau_{org} = (C, \{?org\}, \Omega)$, where

$$C = Compl((?c, lang, ?lang), (?org, founder, ?c))$$

and $\Omega = \{\{?org \mapsto org_1\}, \{?org \mapsto org_2\}, \{?org \mapsto org_3\}\}$. It is the case that $[\![P_{\tau_{org}}]\!]_{(\tilde{P}_{usa}, G_{org})} = \{\{\, ?c \mapsto usa, ?lang \mapsto \tilde{l}, ?org \mapsto org_2 \,\}\}$, where $P_{\tau_{org}}$ is the BGP of the statement C of the template τ_{org}.

Next, in the prioritized evaluation of a BGP P_τ over $(\tilde{P}, G)$, we apply a pruning technique based on the following observation. Each answer mapping $\mu \in [\![P_\tau]\!]_{(\tilde{P},G)}$ determines a nonempty subset $P'_\tau \subseteq P_\tau$ such that $\mu P'_\tau \subseteq \tilde{P}$ and $\mu P''_\tau \subseteq G$ for its complement $P''_\tau := P_\tau \setminus P'_\tau$. Since frozen variables only occur in $\tilde{P}$ and not in G, we conclude that for every variable $?v$ that occurs both in P'_τ and P''_τ it must be the case that $\mu(?v)$ is not a frozen variable.

The algorithm with pruning proceeds as follows. For each nonempty subset $P'_\tau \subseteq P_\tau$, we first evaluate P'_τ over $\tilde{P}$, which yields partial answers ν. We try to complete each such partial answer ν by evaluating the instantiated complement $\nu(P''_\tau)$ over G and joining the answers resulting from this with ν itself. We prune the answers ν of the first evaluation step by keeping only those mappings for which no term $\nu(?v)$, $?v \in \mathit{var}(P''_\tau)$, is a frozen variable. We call such a ν *pure*. Clearly, for non-pure mappings the subsequent evaluation over G can only result in the empty set. Formally, we compute the union

$$\bigcup_{\substack{P'_\tau \subseteq P_\tau \\ P'_\tau \neq \emptyset}} \; \bigcup_{\substack{\nu \in [\![P'_\tau]\!]_{\tilde{P}} \\ \nu \text{ is pure}}} \{\nu\} \bowtie [\![\nu(P_\tau \setminus P'_\tau)]\!]_G,$$

which equals $[\![P_\tau]\!]_{(\tilde{P},G)}$ as just explained.

We denote the projection of a mapping μ wrt. a set W of variables as $\pi_W(\mu)$. Given a set $\mathcal{T}$ of completeness templates, a frozen BGP $\tilde{P}$, and a graph G, we now define the *template-based transfer operator* $T_\mathcal{T}$ as follows:

$$T_\mathcal{T}(\tilde{P}, G) = \bigcup_{\substack{\tau \in \mathcal{T} \\ \tau = (C, V_\tau, \Omega)}} \{\mu P_\tau \mid \mu \in [\![P_\tau]\!]_{(\tilde{P},G)} \text{ and } \pi_{V_\tau}(\mu) \in \Omega\}.$$

The above operator computes for each template τ the prioritized evaluation of the BGP P_τ over $(\tilde{P}, G)$, keeps only those mappings compatible with Ω, and then takes the union. The crucial point here is that we first evaluate the BGP of the template, and only after that we check which answers correspond to instantiations by Ω. By the definition of completeness templates and the prioritized evaluation of BGPs, it is the case that P_0 as in Eq. (4.2) can alternatively be computed using $T_\mathcal{T}$, as stated in Proposition 4.3.

Proposition 4.3. *Given a BGP P, a graph G, and a set $\mathcal{T}$ of completeness templates, it is the case that*

$$P_0 = P \cap \tilde{id}^{-1}(T_{\mathbf{C}_\mathcal{T}}(\tilde{P} \cup G)) = P \cap \tilde{id}^{-1}(T_\mathcal{T}(\tilde{P}, G)).$$

Partial Matching. As there can be many completeness templates, we want to rule out the irrelevant ones, that is, those templates that do not contribute to query completeness. Basically, they are the templates with no overlapping triple patterns (modulo variable generalization) over the query.

Let us first sketch the idea of partial matching. Here, we rely on hashmaps. We use each triple pattern of a template as a hashkey, by which the template can be retrieved. Thus, a template with three triple patterns, for example, can be retrieved in three different ways. To find templates that are potentially applicable to a frozen BGP $\tilde{P}$, we perform a hashmap lookup for each triple pattern of P and for all possible generalizations of that triple pattern where non-predicate terms are replaced by a variable.

Let us formalize the above idea. Our main goal here is partial matching: retrieving only completeness templates having a triple pattern that can potentially be mapped to a triple in a frozen BGP $\tilde{P}$. To this end, we first introduce a *signature operator* that abstracts away concrete variables by replacing every occurrence of a variable with the reserved IRI `_var`. The *signature* of an element $t \in I \cup L \cup V$ is defined as

$$\sigma(t) = \begin{cases} t, & \text{if } t \in I \cup L \\ \texttt{\_var}, & \text{if } t \in V. \end{cases}$$

The signature of a triple pattern (s, p, o) is defined as $\sigma((s, p, o)) = (\sigma(s), \sigma(p), \sigma(o))$. Furthermore, the signature of a BGP P is defined as $\sigma(P) = \{\, \sigma((s, p, o)) \mid (s, p, o) \in P \,\}$. As an illustration, the signature of the BGP $P_{usa} = \{(usa, lang, ?l)\}$ is as follows: $\sigma(P_{usa}) = \{\, (usa, lang, \texttt{\_var}) \,\}$.

Next, we index completeness templates according to (the signatures of) their triple patterns. For this purpose, we define a mapping M from signature triples to sets of completeness templates such that the signature triple is in the signature of the template's BGP:

$$M((s, p, o)) = \{\, \tau \in \mathcal{T} \mid (s, p, o) \in \sigma(P_\tau) \,\}.$$

In practice, such a mapping can be realized by standard hashmaps, providing fast retrieval operations. Given a signature triple (s, p, o), the *generalization operator* $\texttt{gen}((s, p, o))$ computes the set of all generalizations where non-predicate terms can become variables. As an illustration, the generalization of the signature triple $(\mathit{usa}, \mathit{lang}, \texttt{\_var})$ is the set $\{(\mathit{usa}, \mathit{lang}, \texttt{\_var}), (\texttt{\_var}, \mathit{lang}, \texttt{\_var})\}$.

Now, we are ready to define an operator to get completeness templates that can potentially 'transfer' at least one triple in the frozen BGP $\tilde{P}$. The operator $\texttt{pmatch}(P, \mathcal{T})$ computes the set of *partially matched* completeness templates wrt. P and $\mathcal{T}$, and is defined as

$$\bigcup_{(s,p,o) \in \sigma(P)} \{M((s', p', o')) \mid (s', p', o') \in \texttt{gen}((s, p, o))\}.$$

The operator computes the union of the mapping results over signature generalization of all triple patterns in the BGP P. By the construction of the mapping M and the generalization operator, it is the case that $\texttt{pmatch}(P, \mathcal{T})$ preserves P_0 in Eq. (4.2), as stated in Proposition 4.4.

Proposition 4.4. *Given a BGP P, a graph G, and a set $\mathcal{T}$ of completeness templates, it is the case that*

$$P_0 = P \cap \tilde{id}^{-1}(T_{\mathbf{C}_{\mathcal{T}}}(\tilde{P} \cup G)) = P \cap \tilde{id}^{-1}(T_{\mathbf{C}_{pmatch(P,\mathcal{T})}}(\tilde{P} \cup G)).$$

This means that instead of taking all the templates in $\mathcal{T}$, it is enough to consider only the subset $\texttt{pmatch}(P, \mathcal{T})$, which is potentially much smaller than $\mathcal{T}$.

4.2.2. Experimental Evaluation of Data-aware Completeness Reasoning

Having described our optimization techniques for data-aware completeness reasoning, we now would like to analyze how well the techniques can provide speed-up, in particular wrt. a realistic scenario, and how feasible it is to perform data-aware completeness reasoning at all. This subsection reports on our evaluation of Wikidata-based completeness reasoning experiments. First, we describe our experimental setup, and then discuss the results of the experiments.

Experimental Setup. The reasoning program and experiment framework were implemented in Java using the Apache Jena library.[10] We used the direct-statement fragment (i.e., the fragment with no qualifiers nor references) of Wikidata as our *data graph*, consisting of around 110 mio triples.[11] We chose Wikidata mainly due to its relatively large size, recent popularity, and good quality, making it suitable for our data-aware experiment. The graph was loaded into a Jena TDB triple store.

Our *queries* were generated based on human-made, openly available queries on the Wikidata query page.[12] We extracted the BGPs of the queries and transformed the vocabulary of the queries to the direct statements vocabulary. These BGPs acted as a 'base' for generating our experiment queries: (*i*) for each base, we evaluated it over the Wikidata graph; (*ii*) we took randomly 20 of the result mappings of the base, projected on the first variable of the base;[13] and (*iii*) we generated queries by instantiating the query bases with these projected mappings. The *completeness statements* are generated in a similar way: (*i*) for each base, we evaluated it over the Wikidata graph; (*ii*) from the answer mappings, we took randomly 50% of them, projected to the first variable of the base; and (*iii*) we generated completeness statements by instantiating the base with the respective mappings as the statements' BGPs. In this setting, we also naturally represent completeness statements by completeness templates as follows: we took the base BGP as the template's BGP, and the projected mappings as the template's mappings.

We measured the runtime of completeness reasoning with optimizations and query evaluation. Each measurement was repeated 10 times and we took the median. The experiments were done on a laptop with Intel Core i5 2.50 GHz-processor and 8 GB memory.

Results and Discussion. In the experiments, we observed the query evaluation time and completeness reasoning time from 1,160 queries, with the average query length of 2.58. There were 445,628 completeness statements generated, with the average completeness statement

[10]`http://jena.apache.org/`

[11]`https://tools.wmflabs.org/wikidata-exports/rdf/exports/20160201/`

[12]`https://www.mediawiki.org/w/index.php?title=Wikibase/Indexing/SPARQL_Query_Examples&oldid=2099085`

[13]We imposed some ordering over the triple patterns in the BGPs.

Table 4.3. Average runtime comparison of query evaluation and completeness reasoning grouped by query length, where $|Q|$ is the query length, N_Q is the number of queries, t_Q is the average of query evaluation time, and t_C is the average of completeness reasoning time

$\|Q\|$	N_Q	t_Q	t_C
1	228	2.82 ms	5.43 ms
2	355	1.86 ms	131.51 ms
3	387	2.53 ms	138.22 ms
4	125	1.63 ms	326.45 ms
5	42	1.36 ms	155.45 ms
6	3	2.41 ms	114.26 ms
8	20	1.93 ms	670.66 ms

length (i.e., the number of triple patterns in the BGP of completeness statements) of 2.43. Furthermore, those statements were represented by 66 completeness templates, corresponding to the number of base BGPs to generate the queries. On average, query evaluation took 2.23 ms, whereas completeness reasoning took 140.09 ms, which was still relatively fast.

To get more detailed observations, we broke down the experiment results by query length (as shown in Table 4.3). There is no clear pattern for both query evaluation and completeness reasoning as it is not always the case that the longer the query gets, the longer the runtime becomes. Interestingly though, the completeness reasoning time for queries of length 1 is much faster than the others. This is likely due to smaller partial matches with templates and easier prioritized evaluation in the reasoning, in the sense that processing such queries does not even need to see the data graph whenever the corresponding templates' BGPs are also of length 1 (recall Subsection 4.2.1). Overall, though completeness reasoning is slower than query evaluation, it is still relatively fast in absolute scale (i.e., always below 700 ms). To get an idea of how long plain completeness reasoning is (i.e., without optimizations), we took randomly 10 queries for each query length group and measured the reasoning time. We then computed the average reasoning time with a weighting scheme that respects the query distribution as in Table 4.3. The average reasoning time was 15 s, which is relatively slow. A possible explanation is that for plain completeness reasoning, all the statements were applied repeatedly over the union of the frozen BGP $\tilde{P}$ and the data graph G. We then measured the reasoning time for those queries using only the partial matching technique, where we constructed a single completeness template for every completeness statement. In this case, the average reasoning time was 401.8 ms, as opposed to 140.09 ms, where completeness templates to represent multiple statements were additionally used. Without using templates, partial matching might still get many completeness statements that have to be individually evaluated over $\tilde{P} \cup G$, as opposed to the template's simultaneous processing. This shows that both optimization techniques, that is, completeness templates and partial matching, may help speed up the reasoning.

4.3. Summary

In this chapter, we have provided optimization techniques for the problem of completeness entailment both in the data-agnostic and data-aware settings.

For the data-agnostic setting, we proposed the constant-relevance principle, to reduce the number of completeness statements employed in the reasoning about query completeness. Then, we developed techniques for the retrieval of constant-relevant statements, based on several index structures: standard hashing, inverted indexes, and tries, and performed a comparative performance evaluation over those indexes. Finally, we have experimentally shown that our proposed techniques can improve the feasibility of data-agnostic reasoning.

For the data-aware case, we have developed optimization techniques based on completeness templates and partial matching. Our Wikidata-based experimental evaluation has shown that completeness reasoning with the optimized techniques can be performed relatively fast, taking on average 140.09 ms, much faster than plain completeness reasoning which took around 15 s.

Chapter 5

Soundness Reasoning

As RDF generally follows the open-world assumption, the use of negation in SPARQL queries can lead to unsound answers (as exemplified in Section 1.2). We have proposed completeness statements as metadata specifying that certain kinds of information are entirely recorded in an RDF dataset. In this chapter, we leverage completeness statements to check whether we can guarantee the soundness of SPARQL query answers when negation is used. We distinguish between the soundness of a specific answer of a graph pattern and the soundness of a graph pattern as a whole. We provide a formalization and characterize the problem of soundness checking via reduction to completeness checking. We further conduct an experimental evaluation based on Wikidata, to demonstrate the feasibility of our framework. Partial, preliminary results of this chapter have been published in [32], while the full results have been published in [29].

5.1. SPARQL with Negation

Here, we define SPARQL queries with negation, by extending our definition of the positive fragment of SPARQL as in Section 2.1. We introduce notations that are concise and more convenient for our purposes than the original syntax [46]. Recall that a *basic graph pattern* (BGP) is a set of triple patterns. A ***NOT-EXISTS*** *pattern* is constructed by negating a BGP via '$\neg\exists$'. A *graph pattern* P, as used throughout this chapter, is defined as a set of triple patterns and `NOT-EXISTS` patterns. The *positive part* of P, denoted P^+, consists of all triple patterns in P, and the *negative part* of P, denoted P^-, consists of the BGPs of all `NOT-EXISTS` patterns in P. A mapping μ is a partial function $\mu\colon V \to I \cup L$. The evaluation $[\![P]\!]_G$ of a graph pattern P over a graph G produces a set of mappings and is defined in [46] as:

$$\{\, \mu \in [\![P^+]\!]_G \mid \forall P_i \in P^- \,.\, [\![\mu(P_i)]\!]_G = \emptyset \,\}.$$

We assume that graph patterns are consistent, i.e., $[\![P]\!]_G \neq \emptyset$ for some graph G.

5.2. Motivation and Formalization

We next introduce our two core problems, answer soundness and pattern soundness.

5.2.1. Answer Soundness

Consider the following graph pattern, asking for countries where `en` is no official language and whose official languages (if any) do not include an official language of an EU founder:

$$\begin{aligned} P_l = \{&(?c, a, \mathit{country}), \neg\exists\{\, (?c, \mathit{lang}, \mathit{en}) \,\},\\ &\neg\exists\{\, (?c, \mathit{lang}, ?l), (?f, \mathit{lang}, ?l), (\mathit{EU}, \mathit{founder}, ?f) \,\}\}. \end{aligned}$$

For the sake of example, consider the following graph about countries:

$$\begin{aligned} G_l = \{&(\mathit{ger}, a, \mathit{country}), (\mathit{usa}, a, \mathit{country}), (\mathit{sgp}, a, \mathit{country}),\\ &(\mathit{spa}, a, \mathit{country}), (\mathit{ger}, \mathit{lang}, \mathit{de}), (\mathit{spa}, \mathit{lang}, \mathit{es}),\\ &(\mathit{EU}, \mathit{founder}, \mathit{ger})\}. \end{aligned}$$

For this graph, consider also the set $\mathbf{C}_l$ of the following four completeness statements:

- $C_{ger} = \mathit{Compl}((\mathit{ger}, \mathit{lang}, ?l))$, for all official languages of Germany;
- $C_{usa} = \mathit{Compl}((\mathit{usa}, \mathit{lang}, ?l))$, for all official languages of the USA (i.e., the USA has no official language[1]);
- $C_{spa} = \mathit{Compl}((\mathit{spa}, \mathit{lang}, ?l))$, for all official languages of Spain; and
- $C_{eu} = \mathit{Compl}((\mathit{EU}, \mathit{founder}, ?f))$, for all EU founders.

[1]As it is the case in reality, see also: `https://www.cia.gov/library/publications/the-world-factbook/geos/us.html`

Note that we do not claim anything about the completeness of the official languages of Singapore.

Evaluating the graph pattern over the graph in the standard way gives

$$[\![P_l]\!]_{G_l} = \{\{?c \mapsto \mathit{usa}\}, \{?c \mapsto \mathit{sgp}\}, \{?c \mapsto \mathit{spa}\}\}.$$

We want to verify whether these answers are sound, that is, whether they cannot have been returned due to possibly incomplete information. This amounts to checking that there is no valid extension of G_l wrt. $\mathbf{C}_l$ over which the answers are not returned. Let us analyze $\{?c \mapsto \mathit{usa}\}$. First, we check if $(\mathit{usa}, \mathit{lang}, \mathit{en})$ is certainly not true. Indeed, since we know by the graph and the statement C_{usa} that the USA has no official language, the $(\mathit{usa}, \mathit{lang}, \mathit{en})$ must not be true. Second, we check if $\{(\mathit{usa}, \mathit{lang}, ?l), (?f, \mathit{lang}, ?l), (\mathit{EU}, \mathit{founder}, ?f)\}$ surely fails. This is clearly the case for the same reason as before, namely that there is no official language of the USA. From this reasoning, we conclude that the answer $\{?c \mapsto \mathit{usa}\}$ is sound.

Next, let us analyze $\{?c \mapsto \mathit{sgp}\}$. We check if $(\mathit{sgp}, \mathit{lang}, \mathit{en})$ is indeed not true, that is, if there is no valid extension where $(\mathit{sgp}, \mathit{lang}, \mathit{en})$ is true. Now we have a problem: due to the lack of completeness information, it might be that in reality, `en` is an official language of Singapore, but the fact is missing in our data. Thus, we cannot guarantee the soundness of the answer $\{?c \mapsto \mathit{sgp}\}$.

Last, let us analyze $\{?c \mapsto \mathit{spa}\}$. First, we check if $(\mathit{spa}, \mathit{lang}, \mathit{en})$ is not true. Since we know by the statement C_{spa} and the graph that Spain's official language is only `es`, then $(\mathit{spa}, \mathit{lang}, \mathit{en})$ must not be true. Second, we check if the following BGP,

$$\{(\mathit{spa}, \mathit{lang}, ?l), (?f, \mathit{lang}, ?l), (\mathit{EU}, \mathit{founder}, ?f)\},$$

evaluates to false. From the graph and the statements C_{ger} and C_{eu}, we know that `de` is the only official language of Germany as the only EU founder, which is different from `es`. Thus, the pattern must evaluate to false. We conclude that the answer $\{?c \mapsto \mathit{spa}\}$ is sound.

In summary, given answers of a graph pattern over a graph with completeness statements, we have reasoned by case analysis whether each answer is sound.

5.2.2. Pattern Soundness

Consider now the following graph pattern asking for countries where `en` is no official language and that are not EU founders:

$$\begin{aligned} P_f = \{(?c, a, \mathit{country}), \neg\exists\{\,(?c, \mathit{lang}, \mathit{en})\,\},\\ \neg\exists\{\,(\mathit{EU}, \mathit{founder}, ?c)\,\}\}. \end{aligned}$$

Consider also the set $\mathbf{C}_f$ of two completeness statements:

- $C_{lang} = \mathit{Compl}((?c, a, \mathit{country}), (?c, \mathit{lang}, ?l))$, for all languages of countries and
- $C_{eu} = \mathit{Compl}((\mathit{EU}, \mathit{founder}, ?f))$, for all EU founders.

It is actually the case that the statements guarantee the soundness of the graph pattern P_f alone, i.e., all answers returned by P_f are sound, independently of the queried graph. In other words, given P_f and $\mathbf{C}_f$, the soundness of all answers is guaranteed for any possible graph, even with totally different languages and EU founders. Let us see why. Consider an arbitrary graph G and suppose the pattern evaluation over G returns an answer $\{?c \mapsto \tilde{c}\}$ for an IRI $\tilde{c}$. To be sure that $\{?c \mapsto \tilde{c}\}$ is sound, we must make sure that $\tilde{c}$ does not have `en` as an official language and is not an EU founder. By the statement C_{lang}, it is the case that G is complete for all languages of countries. Therefore, G is also complete for all languages of $\tilde{c}$. The fact that $\tilde{c}$ is returned means that `en` is not among its official languages. Now, due to C_{eu}, it is the case that G is complete for all EU founders. Again, the fact that $\tilde{c}$ is returned means that $\tilde{c}$ is not an EU founders. Thus we can be sure that the answer $\{?c \mapsto \tilde{c}\}$ is sound. Since the answer and the graph were arbitrary, we conclude that the set $\mathbf{C}_f$ of completeness statements entails the soundness of P_f.

In this scenario, as opposed to answer soundness, we have reasoned whether the soundness of an arbitrary answer of a graph pattern over an arbitrary graph can be guaranteed by a set of completeness statements.

5.2.3. Formalization

Let us first formally define what soundness of an answer means. Consider a graph pattern P, a mapping μ, and an extension pair (G, G'). We say that (G, G') entails *the soundness of* μ *for* P, written

Sound(μ, P) if, whenever $\mu \in [\![P]\!]_G$, then it is the case that $\mu \in [\![P]\!]_{G'}$. Note that for $\mu \notin [\![P]\!]_G$, it is trivial that $(G, G') \models$ *Sound*(μ, P). Therefore, we are only interested in the soundness of answers occurring in $[\![P]\!]_G$. Given a set $\mathbf{C}$ of completeness statements, a graph G, a graph pattern P, and a mapping $\mu \in [\![P]\!]_G$, we say that $\mathbf{C}$ *and* G *entail the soundness of the mapping* μ *of* P, written as $\mathbf{C}, G \models$ *Sound*(μ, P), if for all extension pairs $(G, G') \models \mathbf{C}$ it holds that $(G, G') \models$ *Sound*(μ, P). In our motivating scenario we saw that *usa* is a sound answer while *sgp* is not, thus $\mathbf{C}_l, G_l \models$ *Sound*$(\{?c \mapsto usa\}, P_l)$, while $\mathbf{C}_l, G_l \not\models$ *Sound*$(\{?c \mapsto sgp\}, P_l)$.

Now let us define the soundness of a graph pattern as a whole, called pattern soundness. As opposed to answer soundness, here we abstract over *all* possible answers of a graph pattern. For a graph pattern P, *the soundness of* P is expressed as *Sound*(P). Given an extension pair (G, G'), we define that (G, G') *satisfies the soundness of* P, written $(G, G') \models$ *Sound*(P), if $[\![P]\!]_G \subseteq [\![P]\!]_{G'}$. Given a set $\mathbf{C}$ of completeness statements and a graph pattern P, we say that $\mathbf{C}$ *entails the soundness of* P, written as $\mathbf{C} \models$ *Sound*(P), if for all extension pairs $(G, G') \models \mathbf{C}$, it holds that $(G, G') \models$ *Sound*(P). In our motivating scenario, it is the case that $\mathbf{C}_f \models$ *Sound*(P_f).

It follows immediately from the definitions that all answers to a sound pattern are sound.

Proposition 5.1. *Let* $\mathbf{C}$ *be a set of completeness statements and* P *be a graph pattern. Then,* $\mathbf{C} \models$ *Sound*(P) *iff* $\mathbf{C}, G \models$ *Sound*(μ, P) *for every graph* G *and mapping* μ.

5.3. Checking Answer Soundness

In this section, we show how completeness statements over a graph can be used to judge whether an answer obtained by evaluating a graph pattern over the graph is sound. The idea is to reduce the problem of soundness checking to that of completeness checking. Let us first recall the definition of completeness entailment. Given a set $\mathbf{C}$ of completeness statements, a graph G, and a BGP P, the *data-aware completeness entailment* $\mathbf{C}, G \models$ *Compl*(P) is defined as follows: for all extension pairs $(G, G') \models \mathbf{C}$, it holds that $(G, G') \models$ *Compl*(P).

Now, the main theorem of this section intuitively states the following: the soundness of some answer-mapping of a graph pattern over a graph is achieved exactly if all the graph pattern's `NOT-EXISTS`-BGPs, after applying the answer-mapping to them, are complete for the graph.

Theorem 5.2. (Answer Soundness Characterization) *Let G be a graph, $\mathbf{C}$ be a set of completeness statements, P be a graph pattern, and $\mu \in [\![P]\!]_G$ be a mapping. Then, it is the case that*

$$\mathbf{C}, G \models \mathit{Sound}(\mu, P) \text{ iff for all } P_i \in P^-.\ \mathbf{C}, G \models \mathit{Compl}(\mu P_i).$$

Proof. ($\Leftarrow$) Let $\mu \in [\![P]\!]_G$ be a mapping. Suppose that for all $P_i \in P^-$, we have $\mathbf{C}, G \models \mathit{Compl}(\mu P_i)$. Take an extension pair (G, G') satisfying $\mathbf{C}$. We will show that $\mu \in [\![P]\!]_{G'}$. Since $\mu \in [\![P]\!]_G$ and $G \subseteq G'$, it holds that $\mu \in [\![P^+]\!]_{G'}$. It is left to show that for all $P_i \in P^-$, we have $[\![\mu P_i]\!]_{G'} = \emptyset$. Take an arbitrary $P_i \in P^-$. The inclusion $[\![\mu P_i]\!]_{G'} \subseteq [\![\mu P_i]\!]_G$ holds because $\mathbf{C}, G \models \mathit{Compl}(\mu P_i)$. Moreover, the equality $[\![\mu P_i]\!]_G = \emptyset$ holds because $\mu \in [\![P]\!]_G$. Hence, it is the case that $[\![\mu P_i]\!]_{G'} = \emptyset$.
($\Rightarrow$) We give a proof by contrapositive. Suppose there is a BGP $P_w \in P^-$ ('w' for witness) such that $\mathbf{C}, G \not\models \mathit{Compl}(\mu P_w)$. We will show that $\mathbf{C}, G \not\models \mathit{Sound}(\mu, P)$. Since it is the case that $\mathbf{C}, G \not\models \mathit{Compl}(\mu P_w)$, there must be a mapping ν such that: (i) $\mathit{dom}(\nu) = \mathit{var}(\mu P_w)$; (ii) $(G, G \cup \nu\mu P_w) \models \mathbf{C}$; and (iii) $\nu\mu P_w \nsubseteq G$. This implies that $\nu \notin [\![\mu P_w]\!]_G$ and $\nu \in [\![\mu P_w]\!]_{G \cup \nu\mu P_w}$. Now, we will show that $(G, G \cup \nu\mu P_w) \not\models \mathit{Sound}(\mu, P)$. Since $\nu \in [\![\mu P_w]\!]_{G \cup \nu\mu P_w}$, it holds that $\mu \notin [\![P]\!]_{G \cup \nu\mu P_w}$. On the other hand, it is the case that $\mu \in [\![P]\!]_G$ from our assumption. Thus, $(G, G \cup \nu\mu P_w) \not\models \mathit{Sound}(\mu, P)$. □

Example 5.3. Consider the motivating scenario of answer soundness. Take the mapping $\{?c \mapsto \mathit{usa}\} \in [\![P_l]\!]_{G_l}$. Both the entailment $\mathbf{C}_l, G_l \models \mathit{Compl}((\mathit{usa}, \mathit{lang}, \mathit{en}))$ and the entailment

$$\mathbf{C}_l, G_l \models \mathit{Compl}((\mathit{usa}, \mathit{lang}, ?l), (?f, \mathit{lang}, ?l), (\mathit{EU}, \mathit{founder}, ?f))$$

hold. By Theorem 5.2, it is the case that $\mathbf{C}_l, G_l \models \mathit{Sound}(\{?c \mapsto \mathit{usa}\}, P_l)$.

In contrast, take the mapping $\{?c \mapsto \mathit{sgp}\} \in [\![P_l]\!]_{G_l}$. It is the case that $\mathbf{C}_l, G_l \not\models \mathit{Compl}((\mathit{sgp}, \mathit{lang}, \mathit{en}))$ with the extension pair $(G_l, G_l \cup \{(\mathit{sgp}, \mathit{lang}, \mathit{en})\})$ as a counterexample. Thus, it holds that $\mathbf{C}_l, G_l \not\models \mathit{Sound}(\{?c \mapsto \mathit{sgp}\}, P_l)$.

In fact, Theorem 5.2 holds for a wider class of graph patterns than defined in this chapter. We only need that the positive part of the pattern be monotonic, that is, a mapping remains a solution over all extensions of the graph G. We do not make this formal to keep the exposition simple.

Complexity. From Theorem 5.2, the check whether an answer is sound wrt. a set of completeness statements and a graph can be reduced to a linear number of data-aware completeness checks (as discussed in Chapter 3). From this, it follows that the complexity of the answer soundness entailment problem is in Π_2^P. Moreover, the answer soundness problem is also Π_2^P-hard as the completeness problem can be reduced to it by using Theorem 5.2. Nevertheless, from a practical perspective, one may expect graph patterns (including BGPs used to construct completeness statements) to be short, giving us a potentially manageable answer soundness check. Section 5.5 reports an experimental study of answer soundness checking in practical settings.

5.4. Checking Pattern Soundness

As demonstrated in our motivating scenario, it might be the case that completeness statements guarantee the soundness of a graph pattern as such, that is, all answers returned by the graph pattern are known to be sound, no matter the specifics of the graph. To characterize pattern soundness, we follow the same strategy as before: we reduce the problem of soundness checking to completeness checking.

First, we generalize completeness statements to *conditional completeness statements*, which express the completeness of a BGP under the condition of another BGP. Given two BGPs P and P', *the completeness of P wrt. P'* is denoted as $\mathit{Compl}(P \mid P')$. Given an extension pair (G, G'), we define that $(G, G') \models \mathit{Compl}(P \mid P')$ if $[\![(\mathit{var}(P), P \cup P')]\!]_{G'} \subseteq_s [\![P]\!]_G$.[2] This means that the conditional completeness statement is satisfied by the extension pair, whenever the evaluation of the BGP P over the graph G contains the evaluation of P under the condition of P' over the graph G'. For example, the conditional completeness statement $\mathit{Compl}((?c, \mathit{lang}, \mathit{en}) \mid (?c, a, \mathit{country}))$ denotes the completeness of all things having English as their language, provided that the things are of type country. Note

[2]We use '$\subseteq_s$' for set inclusion.

that conditional completeness statements are more general than completeness statements as introduced in Section 2.2, since a completeness statement $Compl(P)$ can be expressed as a conditional completeness statement with the empty condition $Compl(P \mid \emptyset)$. We define that the entailment $\mathbf{C} \models Compl(P \mid P')$ holds if for all extension pairs (G, G') satisfying $\mathbf{C}$, it is the case that $(G, G') \models Compl(P \mid P')$. The following proposition states that such entailment holds iff the $T_{\mathbf{C}}$ application over the prototypical graph $\tilde{P} \cup \tilde{P}'$ includes $\tilde{P}$. Recall that the prototypical graph represents any possible graph that satisfies a BGP.

Proposition 5.4. *For a set* $\mathbf{C}$ *of completeness statements and BGPs* P *and* P'*, it is the case that*

$$\mathbf{C} \models Compl(P \mid P') \text{ iff } \tilde{P} \subseteq T_{\mathbf{C}}(\tilde{P} \cup \tilde{P}').$$

Proof. ($\Rightarrow$) Suppose that $\mathbf{C} \models Compl(P \mid P')$. By definition of the entailment, for all $(G, G') \models \mathbf{C}$, the inclusion $[\![(var(P), P \cup P')]\!]_{G'} \subseteq_s [\![P]\!]_G$ holds. Consider the extension pair (G, G') where $G = T_{\mathbf{C}}(\tilde{P} \cup \tilde{P}')$ and $G' = \tilde{P} \cup \tilde{P}'$. By construction, $(G, G') \models \mathbf{C}$ holds. From our assumption, it follows that $[\![(var(P), P \cup P')]\!]_{G'} \subseteq_s [\![P]\!]_G$. By construction, we have that $\pi_{var(P)}(\tilde{id}) \in [\![(var(P), P \cup P')]\!]_{G'}$ where $\tilde{id}$ is the freeze mapping of the BGP $P \cup P'$ (as defined in Section 2.1). From the set inclusion, it follows that $\pi_{var(P)}(\tilde{id}) \in [\![P]\!]_G$. This implies that $\pi_{var(P)}(\tilde{id})P = \tilde{P} \subseteq G = T_{\mathbf{C}}(\tilde{P} \cup \tilde{P}')$.
($\Leftarrow$) Assume $\tilde{P} \subseteq T_{\mathbf{C}}(\tilde{P} \cup \tilde{P}')$. By this assumption and the prototypicality of $\tilde{P} \cup \tilde{P}'$, which represents any possible graph satisfying $P \cup P'$, it is the case that $\mathbf{C} \models Compl(P \mid P')$. □

In the motivating scenario of pattern soundness, it holds that $\mathbf{C}_f \models Compl((?c, lang, en) \mid (?c, a, country))$ due to the following inclusions:

- $\{(\tilde{c}, lang, en)\} \subseteq \{(\tilde{c}, lang, en), (\tilde{c}, a, country)\}$, and
- $\{(\tilde{c}, lang, en), (\tilde{c}, a, country)\} \subseteq T_{\mathbf{C}_f}(\{(\tilde{c}, lang, en), (\tilde{c}, a, country)\})$.

This means that the set $\mathbf{C}_f$ of statements guarantees the completeness of all things whose official language is English, under the condition that those things are of type country.

The following lemma states that the soundness of a graph pattern can be guaranteed if each BGP of the `NOT-EXISTS` patterns is complete under the condition of the positive part of the graph pattern.

Lemma 5.5. *Given a set* $\mathbf{C}$ *of completeness statements and a graph pattern* P*, it is the case that*

$$\mathbf{C} \models \mathit{Sound}(P) \quad \text{if} \quad \text{for all } P_i \in P^- \,.\, \mathbf{C} \models \mathit{Compl}(P_i \mid P^+).$$

Proof. Assume that for all $P_i \in P^-$, it is the case that $\mathbf{C} \models \mathit{Compl}(P_i \mid P^+)$. Take any extension pair $(G, G') \models \mathbf{C}$ and suppose there is a mapping $\mu \in [\![P]\!]_G$. We want to show that $\mu \in [\![P]\!]_{G'}$. By $G \subseteq G'$, it holds that $\mu \in [\![P^+]\!]_{G'}$. Thus, it is left to show that for all $P_i \in P^-$, it is the case that $[\![\mu P_i]\!]_{G'} = \emptyset$.

Take any negation part P_i. By $\mathbf{C} \models \mathit{Compl}(P_i \mid P^+)$ and $(G, G') \models \mathbf{C}$, it is the case that $(G, G') \models \mathit{Compl}(P_i \mid P^+)$. Consequently, by $[\![(\mathit{var}(P_i), P_i \cup P^+)]\!]_{G'} \subseteq_s [\![P_i]\!]_G$ and $[\![\mu P_i]\!]_G = \emptyset$, it must be the case that $[\![\mu P_i]\!]_{G'} = \emptyset$. As P_i was arbitrary, it is the case that $\mu \in [\![P]\!]_{G'}$. □

One might wonder whether the converse of the above lemma also holds. However, the following counterexample shows that it does not.

Example 5.6. Consider the following graph patterns:

- $P_1 = \{(?c, a, \mathit{country}), \neg\exists\{(?c, \mathit{lang}, \mathit{en})\},$
  $\neg\exists\{(?c, \mathit{lang}, \mathit{en}), (?c, \mathit{lang}, \mathit{fr})\}\}$
- $P_2 = \{(?c, a, \mathit{country}), \neg\exists\{(?c, \mathit{lang}, \mathit{en}), (?c, \mathit{lang}, ?l)\}\}$

Consider also the singleton set $\mathbf{C} = \{\mathit{Compl}((?c, \mathit{lang}, \mathit{en}))\}$. It is the case that $\mathbf{C} \models \mathit{Sound}(P_1)$ and $\mathbf{C} \models \mathit{Sound}(P_2)$ despite the violation of the right-hand side of Lemma 5.5.

Taking a closer look, one notices that both graph patterns in fact contain redundancies, which can be checked via query containment under set semantics (written $\sqsubseteq_s$). For P_1, the second `NOT-EXISTS` pattern is superfluous due to the first one being more general; whereas for P_2, the triple pattern $(?c, \mathit{lang}, ?l)$ is superfluous since the emptiness of the BGP of the `NOT-EXISTS` pattern only depends on the triple pattern $(?c, \mathit{lang}, \mathit{en})$. Consequently, for both cases having only the statement $\mathit{Compl}((?c, \mathit{lang}, \mathit{en}))$ is sufficient to guarantee their soundness.

To avoid such redundancies, we propose a normal form for graph patterns, called *Non-Redundant Form (NRF)*. A graph pattern P is in NRF if it satisfies that: there is no containment between any distinct BGPs of the negative parts; and every single BGP of the negative parts is minimal. This can be formalized as follows:

- *No redundant negations:* for any distinct BGPs $P_i, P_j \in P^-$, it is the case that:

$$(var(P^+), P^+ \cup P_i) \not\sqsubseteq_s (var(P^+), P^+ \cup P_j).$$

- *No redundant parts in a negation:* for every $P_i \in P^-$, there is no non-empty $P_i' \subset P_i$ such that:

$$(var(P^+), P^+ \cup P_i') \sqsubseteq_s (var(P^+), P^+ \cup P_i).$$

A non-NRF graph pattern can be transformed into an equivalent NRF graph pattern with a polynomial number of NP-checks, by repeating the containment check and redundant part removal until the two conditions above are satisfied. As graph patterns tend to be relatively small in practice, we expect that such a transformation is feasible.

With this notion in place, we can obtain the main theorem of this section. The theorem states that given an NRF graph pattern, the check whether it is sound can be reduced to the check whether each BGP of the `NOT-EXISTS` patterns is complete under the condition of the positive part. Thus, the theorem ensures that the converse of Lemma 5.5 holds for NRF graph patterns.

Theorem 5.7. (PATTERN SOUNDNESS CHARACTERIZATION) *Given a set* $\mathbf{C}$ *of completeness statements and a graph pattern* P *in Non-Redundant Form (NRF), it is the case that*

$$\mathbf{C} \models \mathit{Sound}(P) \text{ iff for all } P_i \in P^- .\ \mathbf{C} \models \mathit{Compl}(P_i \mid P^+).$$

Proof. ($\Leftarrow$) This is a direct consequence of Lemma 5.5.
($\Rightarrow$) We give a proof by contrapositive. Suppose there is a BGP $P_w \in P^-$ ('w' for witness) such that $\mathbf{C} \not\models \mathit{Compl}(P_w \mid P^+)$. By Proposition 5.4, it is the case that $\tilde{P}_w \not\subseteq T_{\mathbf{C}}(\tilde{P}_w \cup \tilde{P}^+)$. Let us prove that for the extension pair $(G, G') = (\tilde{P}^+ \cup T_{\mathbf{C}}(\tilde{P}_w \cup \tilde{P}^+), \tilde{P}_w \cup \tilde{P}^+)$, it is the case that $(G, G') \models \mathbf{C}$, but $(G, G') \not\models \mathit{Sound}(P)$.

By the definition of $T_{\mathbf{C}}$, it holds that $(G, G') \models \mathbf{C}$. We now have to show that $(G, G') \not\models \mathit{Sound}(P)$. By construction, $\tilde{id} \notin [\![P]\!]_{\tilde{P}_w \cup \tilde{P}^+} = [\![P]\!]_{G'}$ where $\tilde{id}$ is the freeze mapping wrt. P^+. We will show that $\tilde{id} \in [\![P]\!]_{\tilde{P}^+ \cup T_{\mathbf{C}}(\tilde{P}_w \cup \tilde{P}^+)} = [\![P]\!]_G$.

By construction, $\tilde{id} \in [\![P^+]\!]_{\tilde{P}^+ \cup T_{\mathbf{C}}(\tilde{P}_w \cup \tilde{P}^+)}$. Thus, it is left to show that for every BGP $P_i \in P^-$, it is the case $[\![\tilde{id} P_i]\!]_{\tilde{P}^+ \cup T_{\mathbf{C}}(\tilde{P}_w \cup \tilde{P}^+)} =$

$\emptyset$. Due to the consistency of P and the non-containment property between different negation parts from the 'no redundant negations' condition of an NRF graph pattern, there is no negation part $P_j \neq P_w$ such that:

$$[\![\tilde{id} P_j]\!]_{\tilde{P}^+ \cup T_{\mathbf{C}}(\tilde{P}_w \cup \tilde{P}^+)} \neq \emptyset.$$

Now, it is left to show that for the BGP P_w, it also holds

$$[\![\tilde{id} P_w]\!]_{\tilde{P}^+ \cup T_{\mathbf{C}}(\tilde{P}_w \cup \tilde{P}^+)} = \emptyset.$$

However, this holds from the consistency of P and the minimality property from the 'no redundant parts in a negation' condition of an NRF graph pattern. Thus, we have shown that $\tilde{id} \notin [\![P]\!]_{G'}$ but $\tilde{id} \in [\![P]\!]_G$, serving as a counterexample for $(G, G') \models \mathit{Sound}(P)$. $\square$

Example 5.8. In the motivating scenario of pattern soundness, it holds that $\mathbf{C}_f \models \mathit{Compl}((?c, \mathit{lang}, \mathit{en}) \mid (?c, a, \mathit{country}))$ and also

$$\mathbf{C}_f \models \mathit{Compl}((\mathit{EU}, \mathit{founder}, ?c) \mid (?c, a, \mathit{country})).$$

By Theorem 5.7, it is the case $\mathbf{C}_f \models \mathit{Sound}(P_f)$.

Complexity. From Theorem 5.7 and Proposition 5.4, it follows that the check whether a graph pattern is sound can be reduced to a linear number of $T_{\mathbf{C}}$ applications, which are basically evaluations of conjunctive `CONSTRUCT` queries. Hence, deciding whether a graph pattern is sound wrt. a set of completeness statements is in NP (and also NP-hard, as checking completeness can also be reduced to checking soundness). From a practical viewpoint, one may expect graph patterns of queries and BGPs of completeness statements to be short, potentially allowing for a feasible soundness check. As in our optimization of data-agnostic completeness checking that uses the constant-relevance principle, our $T_{\mathbf{C}}$ applications here for conditional completeness statements can immediately adopt the principle by indexing all constants appearing in the whole body of the conditional statements. Section 5.5 reports an experimental investigation of pattern soundness checking in practical cases.

Soundness of Queries with Projections. One may wonder whether our characterization here can also be used for queries with negation that involve projections. The next example shows that in general, the condition from Theorem 5.7 is not a necessary condition for pattern soundness entailment of queries with projection.

Example 5.9. Consider the following boolean query, which asks whether it is impossible to right-shift any triple:

$$Q = (\{\}, \{(?x, ?y, ?z), \neg\exists\{\,(?z, ?x, ?y)\,\}\}).$$

Consider also the singleton set of a completeness statement of three possible shifts of triples:

$$\mathbf{C} = \{\mathit{Compl}((?x, ?y, ?z), (?z, ?x, ?y), (?y, ?z, ?x))\}.$$

By case analysis over the availability of triple shifts, one can show that whenever $(G, G') \models \mathbf{C}$, it holds that all answers of Q over G are contained in those over G', and thus, $\mathbf{C}$ entails the pattern completeness of Q, even though $\mathbf{C}$ does not entail $\mathit{Compl}((?z, ?x, ?y) \mid (?x, ?y, ?z))$.

We do not know a characterization of pattern soundness for queries involving projections. Nevertheless, Theorem 5.7 still gives a sufficient condition for soundness of this case.

Combining Soundness and Completeness Reasoning. A graph pattern with negation can be both sound and complete. Theorem 5.7 characterizes when a graph pattern P in NNF is sound wrt. a set $\mathbf{C}$ of completeness statements. One can show that P is complete if and only if the positive part P^+ is complete. Via both characterizations, we can then check whether a graph pattern is sound and/or complete.

5.5. Experimental Evaluation

From the above characterizations, we are now able to check query soundness by reducing it to query completeness checks. For this reason, we can therefore reuse the optimization techniques of completeness reasoning as described in Chapter 4. More specifically, we reuse the constant-relevance technique for optimizing pattern soundness checking, and the completeness templates and partial matching techniques for optimizing answer soundness checking. In this section, we analyze how soundness reasoning behaves in a realistic scenario, in particular: how feasible it is to perform soundness reasoning, how much speed-up can be gained with the optimization techniques, and how does pattern soundness checking compare to answer soundness checking. This section reports on our experimental evaluation based on Wikidata. First, we describe our experimental setup, and then discuss the results of the experiments.

5.5.1. Experimental Setup

The reasoning program and experiment framework were implemented in Java using the Apache Jena library[3] and are available online.[4] As it was the case for the data-aware completeness reasoning experiment in Subsection 4.2.2, we used the direct statements fragment of Wikidata as our data graph, consisting of around 110 mio triples.[5] The graph was loaded into a Jena TDB triple store.

Queries. Wikidata has openly available, human-created queries which are available online.[6] We took these queries as templates to generate queries with negation. We extracted the BGPs of the queries and transformed the vocabulary of the queries to the direct statements vocabulary. We wanted to have queries with negation of various shapes. For this reason, from the BGPs of the queries we generated different sets of queries with negation, differing in the triple patterns that are negated:

- $\mathbf{Q}_{\texttt{oneTP}}$, the last triple pattern is negated;
- $\mathbf{Q}_{\texttt{oneTPoneTP}}$, the last two triple patterns are independently negated, forming two `NOT-EXISTS` patterns;
- $\mathbf{Q}_{\texttt{twoTPs}}$, the last two triple patterns are negated together, forming one `NOT-EXISTS` pattern; and
- $\mathbf{Q}_{\texttt{threeTPs}}$, the last three triple patterns are negated together, forming one `NOT-EXISTS` pattern.

The number of triple patterns negated was set to at most three, which was reasonable, since most real-world queries are of length up to three [8]. We projected out all variables in the positive part to correspond to graph pattern evaluation.

Completeness Statements. We used two different methods of generating completeness statements depending on whether we wanted to perform either answer soundness or pattern soundness checking.

As for the *generation of statements for answer soundness*, we wanted to perform it in such a way that there will be a variety of sound

[3]`http://jena.apache.org/`

[4]`http://completeness.inf.unibz.it/soundness-experiment/`

[5]`https://tools.wmflabs.org/wikidata-exports/rdf/exports/20160201/`

[6]`https://www.mediawiki.org/w/index.php?title=Wikibase/Indexing/SPARQL_Query_Examples&oldid=2099085`

and possibly unsound answers. So, we generated the statements as follows: (*i*) given a query, we evaluated the query and obtained all the answer-mappings; (*ii*) for 25% of these answer-mappings, we applied them to the BGP of each `NOT-EXISTS` pattern of the query and constructed completeness statements out of these instantiated BGPs. This way, we can guarantee that these 25% answer-mappings are sound, while the remainder mappings are possibly unsound.

In this setting, we can naturally represent completeness statements by completeness templates (see Subsection 4.2.1). We took the BGP of the `NOT-EXISTS` patterns as the templates' BGP and the sound answer mappings as the templates' mappings.

In the particular case of $\mathbf{Q}_{\texttt{twoTPs}}$, however, we also performed an additional way of generating completeness statements, which differs on how we get BGPs for completeness statements: instead of taking the whole instantiated BGP of the `NOT-EXISTS` pattern, we also generated completeness statements *separately per triple pattern* in the instantiated BGP. The first triple pattern[7] in the instantiated BGP was taken as is, and the second was (again) instantiated with the answer-mappings from the evaluation of the first triple pattern over the graph.

For the *generation of statements for checking pattern soundness*, we simply transformed the union of the positive part and each BGP of the `NOT-EXISTS` patterns to a completeness statement.

We had five different cases for our experimental evaluation by combining different query sets and completeness statements:

- `oneTP` is where the last triple pattern is negated;
- `oneTPoneTP` is where the last two triple patterns are independently negated;
- `twoTPsTO` ('TO' for together) is where the last two triple patterns are negated together and the statements are for the whole BGP;
- `twoTPsSE` ('SE' for separate) is where the last two triple patterns are negated together, but the statements are obtained separately per triple pattern; and
- `threeTPsTO` ('TO' for together) is where the last three triple patterns are negated together and the statements are for the whole BGP.

[7]We fixed an ordering.

Table 5.1. The number of statements $|\mathbf{C}|$, and the median of query length $|Q|$, of query answers $|[\![Q]\!]_G|$, of query evaluation time t_Q, of answer soundness checking time t_{AS}, of answer soundness checking time per answer $t_{AS/a}$, and of pattern soundness checking time t_{PS} for different cases. All times are in milliseconds.

Case	$\lvert\mathbf{C}\rvert$	$\lvert Q\rvert$	$\lvert[\![Q]\!]_G\rvert$	t_Q	t_{AS}	$t_{AS/a}$	t_{PS}
`oneTP`	37,769	3	24	14	1.57	0.069	0.19
`oneTPoneTP`	119,462	3	82	47	5.8	0.073	0.37
`twoTPsTO`	126,320	3	180	12.7	43.3	0.27	0.21
`twoTPsSE`	138,705	3	180	12.7	17.3	0.1	0.21
`threeTPsTO`	93,080	4	12,099	114	3,873	0.68	0.23

In each case, to perform answer soundness checking, we did not use the statements generated based on pattern soundness since that would have made all the answers sound. On the other hand, to perform pattern soundness checking, we also used all the statements generated based on answer soundness, as otherwise there would have been too few statements (= the number of queries per case). We measured the runtime of soundness reasoning for both pattern and answer, and also that of query evaluation. For each case, we removed the measurements where the query evaluation returned 0 answers, as answer soundness checking would have become trivial. Each measurement was repeated 10 times and we took the median. Moreover, to get the result summary of each experiment case, we also took the median over the case's results. We used median to avoid the effect of extreme values (that is, some queries returned a large number of results, up to about 120,000 results). The experiments were done on a laptop with Intel Core i7 2.50 GHz-processor and 16 GB memory.

5.5.2. Experimental Results and Discussion

To get an idea of how soundness checking performs *without our optimization techniques,* we ran preliminary experiments to measure the runtime of pattern soundness and answer soundness checking with no optimization of the `twoTPsTO` and `threeTPsTO` cases. Here, we set the timeout to 5 minutes. For pattern soundness checking, the median runtime for the `twoTPsTO` case was about 1.5 s and for the `threeTPsTO` case about 1.2 s. For answer soundness checking, however, we experi-

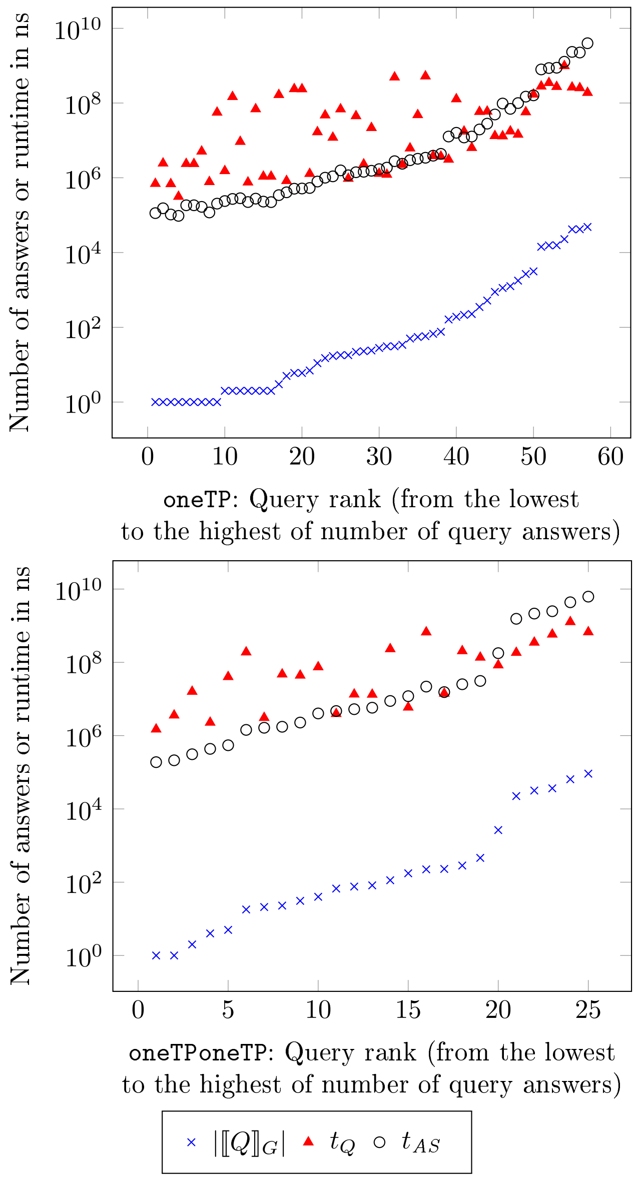


Figure 5.1. Comparison between the number of query answers ($|[\![Q]\!]_G|$), query evaluation time (t_Q), and answer soundness checking time (t_{AS}) for cases `oneTP` and `oneTPoneTP`

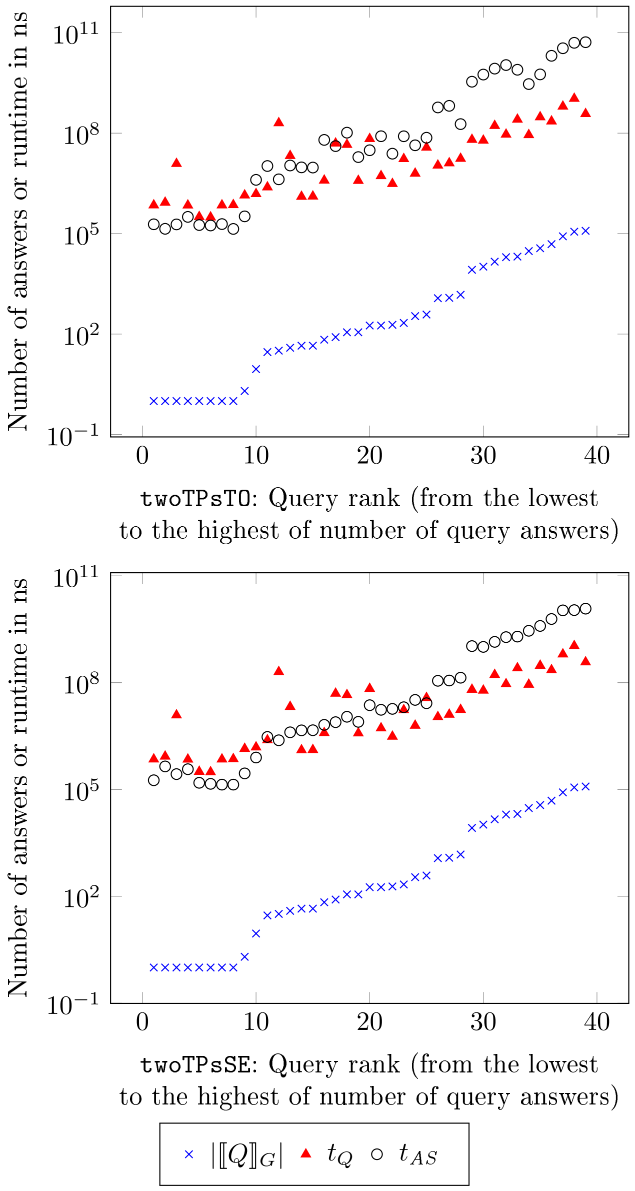


Figure 5.2. Comparison between the number of query answers ($|[\![Q]\!]_G|$), query evaluation time (t_Q), and answer soundness checking time (t_{AS}) for cases `twoTPsTO` and `twoTPsSE`

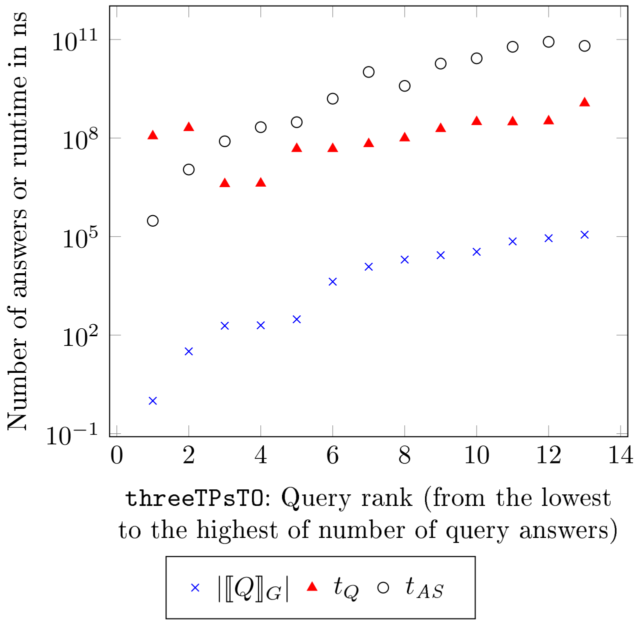


Figure 5.3. Comparison between the number of query answers ($|[\![Q]\!]_G|$), query evaluation time (t_Q), and answer soundness checking time (t_{AS}) for case `threeTPsTO`

enced many timeouts, 22 timeouts out of 39 queries for the `twoTPsTO` case and 11 timeouts out of 13 queries for the `threeTPsTO` case. Timeouts still occurred even when we performed answer soundness checking with partial matching as the only optimization, where we translated each completeness statement trivially into an individual template, without generalization. We experienced 6 timeouts out of 39 queries for the `twoTPsTO` case and 6 timeouts out of 13 queries for the `threeTPsTO` case. This indicates that without the usage of templates, checking answer soundness is hardly feasible.

Now let us see the performance of soundness checking *with all our optimizations.* Table 5.1 summarizes the results of the experiments for all the five cases. Among those cases, the number of statements generated varies, with around 37,000 for Case `oneTP`, and over 93,000 for the others. The median length of queries is either 3 or 4, and the median size of query results varies from around 24 to 12,099. Median query evaluation time ranges from 12 ms to 114 ms.

Median *pattern soundness* checking always takes less than a millisecond, which is more than 1000$\times$ faster than the check without optimization. This is likely due to the fact that pattern soundness checking need not see the data graph, and depends solely on the query and completeness statements. Also, the constant-relevance principle probably helps rule out irrelevant statements before performing the actual check.

As for *answer soundness* checking, we experienced no timeouts, and the runtime is quite comparable with query evaluation time, except for Case `threeTPsTO`. This is possibly due to the large number of answers returned, all of which have to be checked for soundness. This suggests that *with templates* and *partial matching,* answer soundness checking can be done relatively quickly, especially when there are low-to-medium number of query answers. When we break down the time per answer, the computation is less than a millisecond, with the worst case of 0.68 ms for the `threeTPsTO` case. It is likely that the more the triple patterns are in the negation part, the longer the soundness checking per answer is.

Let us look more closely at answer soundness checking. Figures 5.1, 5.2, and 5.3 show the comparison between the number of query answers, query evaluation time, and answer soundness checking time for all the cases. The x-axis is the query order based on the number of query answers in an ascending manner. The y-axis is in log-scale and shows the respective unit (number for the query answers, and ns for the runtime). There is strong evidence of a positive correlation between the number of query answers and the answer soundness checking time. Moreover, we also see the following trend for the first two cases (that is, Figure 5.1): At first, when query answers are not many, query evaluation tends to be slower than answer soundness checking. When the number of query answers increases, the answer soundness checking time outgrows the query evaluation time, for instance, as witnessed by the queries from rank 20 onwards for the case `oneTPoneTP`. For the last three cases, the cross-over point happens earlier than that in the first two cases. This probably has to do with the increasing soundness checking time per answer whenever the number of negated triple patterns increases, as discussed above.

To summarize, we have performed an experiment over a realistic setting based on Wikidata. We have optimized reasoning by representing sets of completeness statements using templates and by using hashmaps to apply only potentially useful statements and templates. As a result, pattern soundness checking can be done quickly, whereas answer soundness checking, though slower than pattern soundness checking, can still be done relatively fast. Moreover, the performance of answer soundness checking positively correlates with the number of query answers. Our optimization techniques have been shown to give a significant speed-up over both reasoning problems. We would also recommend that in practice, before applying answer soundness checking, pattern soundness checking should be done first since it takes less time, and by Proposition 5.1, if pattern soundness holds, then all answers are sound.

5.6. Related Work

The use of negation in querying can be traced back to Codd's relational calculus [23], where a tuple is included in the complement of a relation if not explicitly given in the relation. Reiter [99] and Clark [21] generalized this notion to rule-based systems. They assumed that the failure to find a proof of a fact implies that the negation is true, and called this the *closed-world assumption* (CWA). SPARQL, the standard query language for RDF, supports negation by such a non-existence check [93, 46]. However, since the semantics of RDF imposes the *open-world assumption* (OWA) [49], there remains a *conceptual mismatch* when SPARQL negation is evaluated in a closed-world style. In other words, there is a gap between the normative semantics of negation in SPARQL, and the classical negation ('the negated fact truly holds') [43] due to RDF's openness. The fact that RDF is a positive language, means that one viable way of having negated facts in RDF is by imposing some (partial) completeness assumption over RDF data: whenever P is complete, then all facts not in P are false.

In the Semantic Web, Polleres et al. [88] first observed this mismatch. They proposed to restrict the scope of negation to particular data sources, thus limiting the search for negative information. In their work, no assumption was made as to whether the knowledge in these data sources is complete. Analyti et al. [7] proposed ERDF, an extended RDF that supports negation, as well as derivation rules. ERDF allows one to have local closed-world information via default closure rules for properties and classes. As opposed to their work which considered only a simple partial CWA over atomic classes and properties (e.g., all cars, all child relationships, ...), our work supports more expressive completeness information, thanks to the flexibility of BGPs. From the practical side, negation is featured in test queries of many popular SPARQL benchmarks such as SP^{2}Bench [104], Berlin SPARQL Benchmark (BSBM) [15], and FedBench [103], in which the CWA is employed. As for our work, not only does it provide formalizations, but also optimization techniques for checking the soundness of queries with negation, for which we have shown to improve the feasibility of the checking in Wikidata-based experiment settings.

More recently, Gutierrez et al. [44] proposed an alternative semantics for SPARQL based on certain answers. They argued that the proposed semantics is more suitable to capture RDF peculiarities, such as OWA, unique name assumption (UNA), and blank nodes. For queries with negation, they showed that the queries do not have certain answers, since more facts can be arbitrarily added to falsify the query answers. In our work, we combine between open- and closed-information in RDF, enabling SPARQL queries with negation to have answers that are guaranteed to remain. That is, when queries are guaranteed to be sound by completeness statements, new data that might be added to the graph is restricted by the statements, hence the answers will not be falsified.

5.7. Summary

This chapter introduced the problems of pattern soundness and answer soundness for SPARQL queries involving negation. We have shown how to decide both problems in the presence of completeness information, and provided experimental evidence that our techniques are feasible in realistic settings, where queries and completeness statements are of limited length. While in the presentation we have focused on negation via `NOT EXISTS`, our results apply also to queries with the `MINUS` negation as long as there is a shared variable between the positive part and each of the negative parts.

Our work leaves several issues open. Full characterizations of soundness checking for richer queries that involve selection, union, or arithmetic filter operators are of our interest. In this regard, the current results, which concern the fragment of BGPs with several `NOT EXISTS` patterns, provide a reasonable basis for future investigation. That is, this work already gives sufficient characterizations for richer queries, e.g., queries with selection are sound if the selection-free version of the queries are sound. We also plan to investigate soundness reasoning in the presence of explicit negative information in RDF (e.g., as proposed in [26]). On the practical side, the availability of structured completeness information remains a core issue. We hope that our work provides a further incentive for standardization and data publication efforts in this area, since now not only can completeness statements guarantee query completeness, but also query soundness.

Chapter 6

Time-aware Completeness Reasoning

When creating completeness statements about a data source, one might make the assumption that the data source, regardless of time, is always complete for the parts of data captured by the statements. Indeed, 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 of Reservoir Dogs) or, if the data may still change, the data source has a synchronization mechanism to immediately capture new facts in the real world. However, there might be situations in which such a synchronization is unlikely, like when the data provider is not an authority, or the data originates from crowdsourcing. Consequently, completeness statements can be out-of-date, i.e., the data in the source captured by the statements does not reflect the complete facts in the real world that include new facts. Inspired by natural language completeness statements on Wikipedia, completeness statements can be extended by timestamps. Wikipedia provides a template allowing one to specify that a list is "*complete and up-to-date as of {some specific date}*", as exemplified by the complete list of Twenty-five Year Award winners and Italian DOP cheeses (as previously mentioned in Chapter 1). In this chapter, we discuss how to extend completeness statements to cope with data dynamicity over time, and reason about query completeness given such time-extended statements. The results in this chapter have been published in [28].

6.1. Motivating Scenario

To deal with data dynamicity, a time extension to completeness statements is a necessity. Here, by dynamicity we refer to any addition of data, that is, new information is added without invalidating old information. 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 the statement is likely to grow. However, suppose that the data source fails to capture an update of the data. What then happens 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, i.e., 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 cannot grow anymore. We call such a statement an *unbounded completeness statement.*

Indeed, 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 we also have another statement "Cast of Tarantino movies up to 2016 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 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 the 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 2016 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 wrt. any date, and thus, there is no guaranteed completeness date for the query.

6.2. Time-extended Completeness Framework

We now formalize the extended completeness framework and its semantics. We define a *date* as an element $d \in \mathbb{N} \cup \{ \infty \}$.[1] 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 6.1 (Timestamped Completeness Statement). A *timestamped completeness statement* is of the form

$$\hat{C} = Compl(P_{\hat{C}}, d),$$

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

[1]W.l.o.g. our framework also supports continuous domains (e.g., $\mathbb{R}$), given that the discretization of all known timestamps gives again a discrete space.

Example 6.2. Consider the statements "Crew of Tarantino movies up to 2012 are complete," "Cast of Tarantino movies up to 2016 are complete" and "Movies starring Chaplin are complete and there will not be any updates" as above. They can be represented formally as:

$$\hat{C}_{crew} = \mathit{Compl}(\{\, (?m, \mathit{crew}, ?c), (?m, a, \mathit{TarantinoMov}) \,\}, 2012)$$
$$\hat{C}_{cast} = \mathit{Compl}(\{\, (?m, \mathit{cast}, ?c), (?m, a, \mathit{TarantinoMov}) \,\}, 2016)$$
$$\hat{C}_{chap} = \mathit{Compl}(\{\, (?m, a, \mathit{ChaplinMov}) \,\}, \infty)$$

To select timestamped completeness statements based on their dates, we define the selection $\hat{\mathbf{C}}_{\geq d}$ as

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

The selection $\hat{\mathbf{C}}_{=d}$ is defined analogously. As before, we associate to a statement $\hat{C}$, the `CONSTRUCT` query $Q_{\hat{C}} = (P_{\hat{C}}, P_{\hat{C}})$. Over a graph G, the transfer operator $T_{\hat{\mathbf{C}}}(G)$ is defined similarly to that in Eq. (2.3) in Section 2.2 where we take the union of the results of the evaluation $[\![Q_{\hat{C}}]\!]_G$ of all $\hat{C} \in \hat{\mathbf{C}}$.

RDF Representation. To represent timestamped completeness statements in RDF, we propose to use the datatype representation from the XML Schema Definition (XSD) namespace to represent non-infinity dates, which can also be of various granularities such as years and calendar dates.[2] To represent the infinity, we introduce in our vocabulary[3] the term `infinity`. We also create the property `hasTimestamp` that links 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 a graph and a sequence of graphs, of the form

$$\mathcal{S} = (G_{now}, (G'_1, G'_2, \ldots, G'_{now}, \ldots)),$$

such that (G_{now}, G'_{now}) is an extension pair and it holds that $G'_d \subseteq G'_{d+1}$ for all pairs G'_d, G'_{d+1} in $\mathcal{S}$. We have one base graph only (i.e., G_{now}) to reflect the state of the available graph we have now. On the other hand, we have a sequence of extensions to represent data dynamicity over time wrt. the real world.

[2] `http://www.w3.org/2001/XMLSchema`

[3] `http://completeness.inf.unibz.it/ns`

Example 6.3 (Incomplete Data Series)**.** Let $now = 2016$ and

$$\mathcal{S}_{mov} = (G_{now}, (G'_1, \dots, G'_{2012}, \dots, G'_{now}, \dots))$$

be a series about Tarantino and Chaplin movies which can be graphically represented as in Figure 6.1.[4] Note that in this example the set of movies starring Chaplin will not grow anymore (i.e., The Kid) and any other extension G'_k not shown in the figure is defined accordingly.

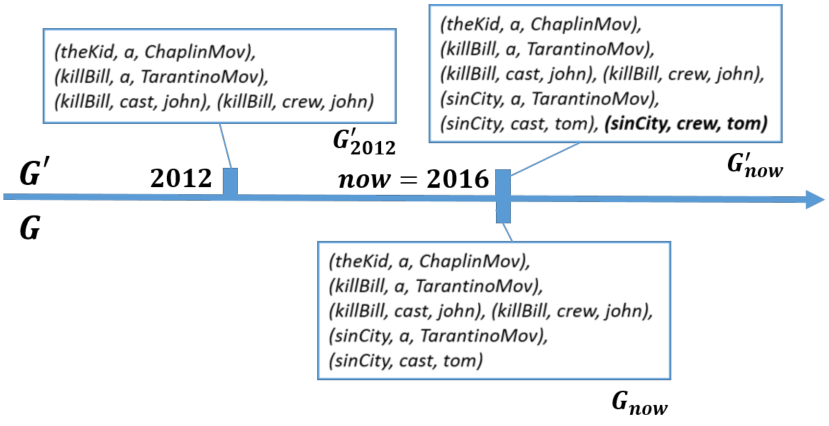

(theKid, a, ChaplinMov),
(killBill, a, TarantinoMov),
(killBill, cast, john), (killBill, crew, john)
G'2012
(theKid, a, ChaplinMov),
(killBill, a, TarantinoMov),
(killBill, cast, john), (killBill, crew, john),
(sinCity, a, TarantinoMov),
(sinCity, cast, tom), (sinCity, crew, tom)
G'now
G'
2012
now = 2016
G
(theKid, a, ChaplinMov),
(killBill, a, TarantinoMov),
(killBill, cast, john), (killBill, crew, john),
(sinCity, a, TarantinoMov),
(sinCity, cast, tom)
Gnow

Figure 6.1. An incomplete data series about Tarantino and Chaplin movies

We now formalize when a series satisfies a timestamped completeness statement. A series $\mathcal{S}$ satisfies a bounded timestamped completeness statement $\hat{C} = Compl(P, d)$, written as $\mathcal{S} \models \hat{C}$, if all the triples constructed by evaluating $Q_{\hat{C}}$ over the extension at date d are in the actual graph, formalized as $[\![Q_{\hat{C}}]\!]_{G'_d} \subseteq G_{now}$. Note that this implies $[\![Q_{\hat{C}}]\!]_{G'_{d'}} \subseteq G_{now}$ for all $d' \leq d$ by the definition of a series. If the statement is unbounded, then the comparison for completeness is made over all extensions: for all $d \in \mathbb{N}$, it must hold that $\mathcal{S} \models Compl(P, d)$. Given a set $\hat{\mathbf{C}}$ of timestamped completeness statements and a series $\mathcal{S}$, we define that $\mathcal{S} \models \hat{\mathbf{C}}$, if for all $\hat{C} \in \hat{\mathbf{C}}$, it holds that $\mathcal{S} \models \hat{C}$.

Example 6.4. Consider the series $\mathcal{S}_{mov}$ in Figure 6.1 and the statements $\hat{C}_{crew}$, $\hat{C}_{cast}$, and $\hat{C}_{chap}$ in Example 6.2. Then, it holds that

[4]For the sake of example, we only use toy data.

$\mathcal{S}_{mov} \models \hat{C}_{crew}$ because it is the case that the result of the evaluation $[\![Q_{\hat{C}_{crew}}]\!]_{G'_{2012}}$, which is the graph,

$$\{\,(\mathit{killBill}, \mathit{crew}, \mathit{john}), (\mathit{killBill}, a, \mathit{TarantinoMov})\,\},$$

is contained in G_{now}. 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}, G'_{2016}, and any other ideal graph G'_k in $\mathcal{S}_{mov}$, the result (i.e., the graph $\{\,(\mathit{theKid}, a, \mathit{ChaplinMov})\,\}$) of the query $Q_{\hat{C}_{chap}}$ evaluated over them is contained in G_{now}.

Query Completeness at a Date. To describe query completeness at date d, we use $\mathit{Compl}(Q, d)$. A series $\mathcal{S}$ satisfies $\mathit{Compl}(Q, d)$ with $d \in \mathbb{N}$, written as $\mathcal{S} \models \mathit{Compl}(Q, d)$, if evaluating Q over the extension at d gives results that are all contained in the results of evaluating Q over the actual graph, formalized as $[\![Q]\!]_{G'_d} \subseteq [\![Q]\!]_{G_{now}}$. Furthermore, a series $\mathcal{S}$ satisfies the unbounded version of query completeness, written as $\mathcal{S} \models \mathit{Compl}(Q, \infty)$, if for all $d \in \mathbb{N}$, it holds that $\mathcal{S} \models \mathit{Compl}(Q, d)$.

Example 6.5. To say that the query asking for all people who were simultaneously cast and crew of Tarantino movies up to 2012 is complete, we can use $\mathit{Compl}(Q_{cc}, 2012)$ where

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

As we can see, $[\![Q_{cc}]\!]_{G'_{2012}}$ returns $(?m \mapsto \mathit{killBill}, ?c \mapsto \mathit{john})$ and is contained in $[\![Q_{cc}]\!]_{G_{now}}$, therefore $\mathcal{S}_{mov} \models \mathit{Compl}(Q_{cc}, 2012)$. On the contrary, $[\![Q_{cc}]\!]_{G'_{2016}}$ returns additionally $(?m \mapsto \mathit{sinCity}, ?c \mapsto \mathit{tom})$, which is not in $[\![Q_{cc}]\!]_{G_{now}}$, therefore $\mathcal{S}_{mov} \not\models \mathit{Compl}(Q_{cc}, 2016)$.

Having defined timestamped completeness statements and query completeness at a date, the question arises as how to actually check the entailment between them. Given a set $\hat{\mathbf{C}}$ of timestamped completeness statements, a query Q, and a date d, we say that $\hat{\mathbf{C}}$ *entails query completeness at* d, written as $\hat{\mathbf{C}} \models \mathit{Compl}(Q, d)$, if for all $\mathcal{S} \models \hat{\mathbf{C}}$, it is the case that $\mathcal{S} \models \mathit{Compl}(Q, d)$. The following lemma gives us a syntactic characterization to decide whether $\hat{\mathbf{C}} \models \mathit{Compl}(Q, d)$. It says that the query completeness at d is entailed by $\hat{\mathbf{C}}$ 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 $\hat{C} \in \hat{\mathbf{C}}$ such that $\mathit{date}(\hat{C}) \geq d$.

Lemma 6.6 (Entailment of Query Completeness at a Date)**.** *Let $\hat{\mathbf{C}}$ be a set of timestamped completeness statements, $Q = (W, P)$ be a query, and d be a date. Then,*

$$\hat{\mathbf{C}} \models \mathit{Compl}(Q, d) \qquad \textit{iff} \qquad \tilde{P} \subseteq T_{\hat{\mathbf{C}}_{\geq d}}(\tilde{P}).$$

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

$$\mathcal{S} = (G_{now}, (\emptyset, \ldots, \emptyset, G'_d, G'_{d+1}, \ldots)),$$

where now is any date such that $now \geq \max(\mathit{date}(\hat{\mathbf{C}}) \setminus \{\, \infty \,\})$, $G_{now} = T_{\hat{\mathbf{C}}_{\geq d}}(\tilde{P})$, and $G'_d = G'_{d+1} = \ldots = \tilde{P}$. By construction, we have that $\mathcal{S} \models \hat{\mathbf{C}}$. However, by the assumption that $\tilde{P} \not\subseteq T_{\hat{\mathbf{C}}_{\geq d}}(\tilde{P})$, it is the case that $[\![Q]\!]_{G'_d} = [\![Q]\!]_{\tilde{P}} \not\subseteq [\![Q]\!]_{T_{\hat{\mathbf{C}}_{\geq d}}(\tilde{P})} = [\![Q]\!]_{G_{now}}$, because the freeze mapping $\tilde{id}$ in $[\![P]\!]_{\tilde{P}}$ is missing in $[\![P]\!]_{T_{\hat{\mathbf{C}}_{\geq d}}(\tilde{P})}$. Therefore, $\mathcal{S} \not\models \mathit{Compl}(Q, d)$.

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

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

Take a series $\mathcal{S} \models \hat{\mathbf{C}}$. We have to show that $\mathcal{S} \models \mathit{Compl}(Q, d)$, that is, $[\![Q]\!]_{G'_d} \subseteq [\![Q]\!]_{G_{now}}$. Suppose there is a mapping $\mu \in [\![Q]\!]_{G'_d}$. Thus, there must be a mapping $\mu_{ext} \supseteq \mu$, where $\mu_{ext} \in [\![P]\!]_{G'_d}$. We will prove that $\mu_{ext} \in [\![P]\!]_{G_{now}}$. By the assumption that $\tilde{P} \subseteq T_{\hat{\mathbf{C}}_{\geq d}}(\tilde{P})$ and the prototypicality of $\tilde{P}$, it holds that $\mu_{ext}\tilde{id}^{-1}(\tilde{P}) \subseteq T_{\hat{\mathbf{C}}_{\geq d}}(\mu_{ext}\tilde{id}^{-1}(\tilde{P}))$. The inclusion can be further extended to:

$$\mu_{ext}\tilde{id}^{-1}(\tilde{P}) \subseteq T_{\hat{\mathbf{C}}_{\geq d}}(\mu_{ext}\tilde{id}^{-1}(\tilde{P})) \subseteq T_{\hat{\mathbf{C}}_{\geq d}}(G'_d),$$

where the last subsumption holds due to $\mu_{ext} \in [\![P]\!]_{G'_d}$. By $\mathcal{S} \models \hat{\mathbf{C}}$, it must be the case that $T_{\hat{\mathbf{C}}_{\geq d}}(G'_d) \subseteq G_{now}$. Therefore, $\mu_{ext}\tilde{id}^{-1}(\tilde{P}) \subseteq G_{now}$, which implies that $\mu_{ext} \in [\![P]\!]_{G_{now}}$.

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

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 wrt. a set $\hat{\mathbf{C}}$ of timestamped completeness statements is the latest date d such that the entailment $\hat{\mathbf{C}} \models \mathit{Compl}(Q, d)$ holds, formally:

$$\mathit{gcd}(Q, \hat{\mathbf{C}}) = \max\{\, d \in \mathbb{N} \cup \{\, \infty \,\} \mid \hat{\mathbf{C}} \models \mathit{Compl}(Q, d) \,\}.$$

We define $\max\{\} = -\infty$, and note that cases where $\mathit{gcd}(Q, \hat{\mathbf{C}}) = -\infty$ correspond to the query Q not having any completeness date.

Example 6.7. Consider the set $\hat{\mathbf{C}} = \{\, \hat{C}_{crew}, \hat{C}_{cast}, \hat{C}_{chap} \,\}$ of completeness statements and the query Q_{cc} as above. It is the case that $\mathit{gcd}(Q_{cc}, \hat{\mathbf{C}}) = 2012$ 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 onwards, however, the query completeness cannot be guaranteed as some crew of Tarantino movies might be missing. Therefore, 2012 is the guaranteed completeness date.

6.3. Computing the Guaranteed Completeness Date

We now analyze how the guaranteed completeness date of a query can be computed. By Lemma 6.6, we can replace the entailment $\hat{\mathbf{C}} \models \mathit{Compl}(Q, d)$ in the definition of the guaranteed completeness date by its syntactic characterization $\tilde{P} \subseteq T_{\hat{\mathbf{C}}_{\geq d}}(\tilde{P})$. In this way, we compute the maximum date from all the dates d in $\hat{\mathbf{C}}$ 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 6.8 (Computing the Guaranteed Completeness Date)**.** *Let $Q = (W, P)$ be a query and $\hat{\mathbf{C}}$ be a set of timestamped completeness statements. Then,*

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

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

Example 6.9. Consider the statements $\hat{\mathbf{C}} = \{ \hat{C}_{crew}, \hat{C}_{cast}, \hat{C}_{chap} \}$ and the query $Q_{cc} = (W_{cc}, P_{cc})$ as above. The set of the dates is $date(\hat{\mathbf{C}}) = \{ 2012, 2014, \infty \}$. Then, we have that:

- $\tilde{P}_{cc} \subseteq \{ (\tilde{m}, cast, \tilde{c}), (\tilde{m}, crew, \tilde{c}), (\tilde{m}, a, TarantinoMov) \}$ $= T_{\hat{\mathbf{C}}_{\geq 2012}}(\tilde{P}_{cc})$
- $\tilde{P}_{cc} \not\subseteq \{ (\tilde{m}, cast, \tilde{c}), (\tilde{m}, a, TarantinoMov) \} = T_{\hat{\mathbf{C}}_{\geq 2014}}(\tilde{P}_{cc})$

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

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

Corollary 6.10 (Complexity of Deciding the Guaranteed Completeness Date)**.** *Deciding whether $gcd(Q, \hat{\mathbf{C}}) \geq d$, given a query Q, a set $\hat{\mathbf{C}}$ of timestamped completeness statements, and a date d, is NP-complete.*

Proof. From Theorem 6.8, there exists an NP procedure to check if $gcd(Q, \hat{\mathbf{C}}) \geq d$: we guess a date $d' \geq d$, and guess the timestamped statements and the mappings over the BGPs of the statements such that $\tilde{P} \subseteq T_{\hat{\mathbf{C}}_{\geq d'}}(\tilde{P})$. It is NP-hard by reduction from the NP-hard problem of data-agnostic completeness entailment (as per Theorem 2.10). □

Algorithm for Finding the Guaranteed Completeness Date. Based on Theorem 6.8, a naive way to compute the guaranteed completeness date is, for every $d \in date(\hat{\mathbf{C}})$, to repeatedly compute $T_{\hat{\mathbf{C}}_{\geq d}}(\tilde{P})$, and then take the maximum of the dates d such that $\tilde{P} \subseteq T_{\hat{\mathbf{C}}_{\geq d}}(\tilde{P})$. This has a drawback since we may be reevaluating the `CONSTRUCT` query of a statement over $\tilde{P}$ several times, though the result is always the same. We could improve the computation by using binary search as $date(\hat{\mathbf{C}})$ has a natural order and $T_{\hat{\mathbf{C}}_{\geq d}}(\tilde{P})$ is monotonic in d. As a consequence, the checking $\tilde{P} \subseteq T_{\hat{\mathbf{C}}_{\geq d}}(\tilde{P})$ would be done only $log(|date(\hat{\mathbf{C}})|)$ times instead of $|date(\hat{\mathbf{C}})|$ times.

Now, we observe the following. For a date d, the result of $T_{\hat{\mathbf{C}}_{\geq d}}(\tilde{P})$ is in fact the union of all $T_{\hat{\mathbf{C}}_{=d'}}(\tilde{P})$ where $d \leq d' \leq max(date(\hat{\mathbf{C}}))$. Consequently, we can compute $T_{\hat{\mathbf{C}}_{\geq d}}(\tilde{P})$ in an incremental way from the latest d'. Thus, we can develop an algorithm to find the guaranteed completeness date where we incrementally compute the union from the latest date in $date(\hat{\mathbf{C}})$ to the earliest date in $date(\hat{\mathbf{C}})$, 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 only needs to be executed at most once over $\tilde{P}$. This means that completeness checking with time is no more complex than completeness checking without time. We formalize this as the algorithm findGCD in Figure 6.2.

ALGORITHM 2: FindGCD

```
    Input: The prototypical graph $\tilde{P}$ of a query, a set of timestamped
           completeness statements $\hat{\mathbf{C}}$
    Output: The guaranteed completeness date $d$
1   $P' \leftarrow \emptyset$
2   $D \leftarrow date(\hat{\mathbf{C}}) \cup \{ -\infty \}$
3   while $\tilde{P} \not\subseteq P'$ and $D \neq \emptyset$ do
4       $d \leftarrow extractMax(D)$
5       $P' \leftarrow P' \cup T_{\hat{\mathbf{C}}_{=d}}(\tilde{P})$
6   end
7   return $d$
```

Figure 6.2. Algorithm for finding the guaranteed completeness date

The algorithm takes as input the prototypical graph $\tilde{P}$ of Q and a set of timestamped completeness statements $\hat{\mathbf{C}}$. At first, we assign the empty set to P', which will store the application results of the transfer operator $T_{\hat{\mathbf{C}}_{=d}}(\tilde{P})$, and assign all the dates in $\hat{\mathbf{C}}$ 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 $extractMax(D)$ to return the latest date d in D and remove it from D. Next, we add to P' the result of $T_{\hat{\mathbf{C}}_{=d}}(\tilde{P})$. At the end of the algorithm, we will return d, which is the guaranteed completeness date of Q wrt. $\hat{\mathbf{C}}$. Note that when $d = -\infty$, the transfer operator $T_{\hat{\mathbf{C}}_{=-\infty}}(\tilde{P})$ would return the empty set, since $\hat{\mathbf{C}}_{=-\infty} = \emptyset$ by definition.

For the algorithm, the following proposition holds, which says that the algorithm is correct, and the `CONSTRUCT` query of a timestamped completeness statement is evaluated at most once in finding the guaranteed completeness date, therefore it is as costly as the standard check of query completeness.

Proposition 6.11. *Let $\hat{\mathbf{C}}$ be a set of timestamped completeness statements and $Q = (W, P)$ be a query. Then,*

- FindGCD*$(\tilde{P}, \hat{\mathbf{C}}) = gcd(Q, \hat{\mathbf{C}})$, and*
- FindGCD*$(\tilde{P}, \hat{\mathbf{C}})$ computes $[\![Q_{\hat{C}}]\!]_{\tilde{P}}$ at most once for every $\hat{C} \in \hat{\mathbf{C}}$.*

6.4. Related Work

Several papers dealt with the incorporation of time into RDF data. Gutierrez et al. [45] was among the first to introduce time annotations over RDF data. They formalized the semantics of temporal RDF graphs, and sketched a temporal query language for RDF. In [66], Lopes et al. developed AnQL, a query language for RDF with annotations, which considers also temporality. Indexing methods for temporal querying were investigated by Pugliese et al. [94], and Tappolet and Bernstein [110]. In our work, RDF graphs are not annotated with timestamps, that is, only completeness statements can be time-aware. As such, a limitation of our work is that it applies only to invariable facts, i.e., facts that hold eternally. The time incorporation into RDF graphs can potentially lift the limitation of our approach. As an illustration, suppose that facts about people being a student are timestamped. Then, we can say that we are complete for all UniBZ students until 2016, in the sense that we have a complete record of people who were UniBZ students from the time until 2016. This would make little sense for triples without timestamps, since say after graduation, people are no longer a student.

Recently, there have been initiatives to combine Linked Data and stream processing. In stream processing, data is produced and queried continuously over time, as opposed to only static data processing. As one of the first RDF stream processors, C-SPARQL [11] used a simple, modular architecture that combines between a SPARQL engine, for dealing with the static part of queries, and a stream processor, for the streaming part of queries. $SPARQL_{Stream}$ [17] focused on ontology-based querying over heterogeneous stream data sources. CQELS [87] concentrated on developing a Linked Data streaming engine from scratch in order to enable low-level optimizations. In the position paper of Keskisärkkä and Blomqvist [58], the authors presented the issue of dynamic boundaries over streams. If events have predictable, uniform boundaries, one can simply set a fixed-size window for every stream query evaluation. However, in some cases, the event boundaries might be unpredictable, e.g., the event might take minutes or hours. Thus, a 'dynamic window' is needed, which is like a window but with adaptable size, in order for query evaluation to have a complete view of the event. Timestamped completeness statements can be potentially leveraged to address this problem: whenever an event is over, a completeness statement can be sent to notify the query processor that data about that event is already streamed completely.

6.5. Summary

In this chapter, we have motivated, formalized, and developed a technique for completeness reasoning with time in the data-agnostic setting. We have introduced timestamped completeness statements, and the guaranteed completeness date, to say that a query is guaranteed to be complete up to a certain point of time. Despite the time addition, time-aware completeness reasoning is no more complex than that without time, in the sense that each timestamped completeness statement is considered at most once in the reasoning. For future work, we plan to develop a technique for time-aware completeness reasoning in the data-aware setting, and also for time-aware soundness reasoning (e.g., the answer 'Arsenal' to the query "Which football club has never won the Champions League?" can be guaranteed to be sound only up to 2017).

Chapter 7

Completeness Management Demonstrators

In previous chapters we have formalized metadata about the partial completeness of RDF data sources. We have also characterized query completeness entailment and developed optimization techniques for checking the entailment. Nevertheless, it is still unclear how systems for managing completeness of RDF data sources can be built. In this chapter, we explore what are the requirements and functionalities of such systems, and demonstrate how such systems can be realized.

In practice, when talking about completeness statements of RDF data sources, we conceive that the statements go through a life cycle. First, completeness statements about RDF data sources are *created*. Next, the created statements are available for *viewing*, so that one can see the completeness state of RDF data sources. Then, completeness statements are *updated* (i.e, edited and deleted), for instance, if they are no longer valid. Finally, completeness statements are used for various *consumption* tasks such as checking query completeness and performing completeness analytics. This completeness life cycle can be illustrated as in Figure 7.1.

The completeness life cycle serves as a basis for developing completeness management systems, which we imagine can be of various types. One possible type is a completeness statement hub. Such a hub stores completeness information across multiple data sources, and supports the check of query completeness based on that completeness information. When query completeness can be guaranteed, the hub can provide a federated rewriting of the query such that parts of the query are evaluated over the data sources that can guarantee their

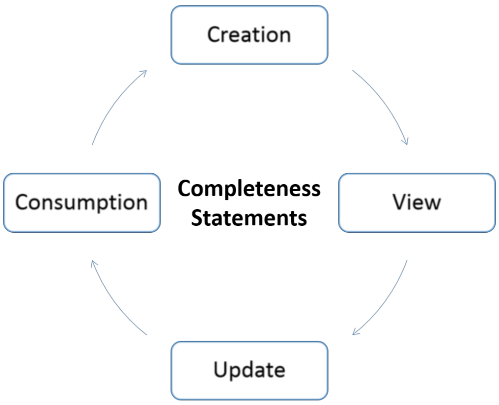


Figure 7.1. Completeness Life Cycle

completeness. Another possibility is a specialized completeness management system for a single knowledge base (KB). For such a system, a tighter connection between completeness statements and the parts of data for which the statements are intended is crucial. Hence, one should be able to directly view which parts of data are annotated with completeness statements. Moreover, data-aware query completeness checking would be more suitable here. Section 7.1 reports CORNER, a demonstrator of a completeness hub, whereas Section 7.2 reports COOL-WD, a demonstrator of a specialized completeness management system for a single KB. For both demonstrators, we focus on practical requirements and functionalities, but not efficiency, which has been investigated in Chapter 4 about optimizations of completeness reasoning. Moreover, both demonstrators support the BGP fragment of SPARQL queries for completeness checking. The description of CORNER has been published in [30] and that of COOL-WD in [90].

7.1. CORNER: A Completeness Reasoner for RDF Data Sources

In this section, we introduce CORNER, a demonstrator of a completeness management hub for RDF data sources. We analyze the practical requirements and describe the architecture of CORNER. Finally, we elaborate the functionalities of CORNER from its user interface (UI). Our system is accessible at `http://corner.inf.unibz.it`.

7.1.1. Motivating Example

In this subsection, we give a motivating example for CORNER. This motivating example is about how a user has a query he wants to answer completely. He therefore has to consult a system with completeness information over multiple data sources. Since multiple data sources are involved, mappings between classes and properties need to be supported by the system. Then, when he asks his query, the system should be able to check the query completeness, and also to suggest a query rewriting where parts of the query are distributed over the complete sources. This motivating example is set up on CORNER so that one can try it out live.

Marty, a moviegoer, is interested in finding all movies starring Quentin Tarantino. This information need can be expressed by the SPARQL BGP query:[1]

```
SELECT * WHERE { ?m actor Tarantino }
```

CORNER has meta-information about parts of LinkedMDB, an RDF data source about movies, and DBpedia, a general purpose RDF data source, that for the sake of example, are supposed to be complete. Completeness statements can be represented in two ways: a human-readable abstract syntax, or an RDF syntax, which implements the abstract syntax. Both syntaxes are accepted by CORNER. Abstract completeness statements have the form $\mathit{Compl}(P_1|P_2)$, consisting of two parts: the *pattern* P_1 and the *condition* P_2. Here we use a general version of completeness statements that may have conditions (as defined in Section 5.4). The completeness statement specifies that the source contains all data with the pattern shape, provided that in addition they satisfy the condition. We argue that completeness statements with conditions are suitable for the multiple data sources scenario since it can be the case that a source is complete under some condition, where the condition is satisfied by some other sources. To express that a source is complete for "all movies starring Tarantino", we write in the abstract syntax

```
Compl(?m actor Tarantino | true).
```

We attach this statement to LinkedMDB but not to DBpedia, since some information that Tarantino was starred in some movies is actually missing in DBpedia. CORNER then analyzes the query and

[1]For simplicity, we omit namespaces.

the statement, and concludes that the query over LinkedMDB can be answered completely, while it cannot give such a guarantee for DBpedia.

Suppose now Marty would also like to see the budget and box-office gross of the movies. This is expressed by the SPARQL BGP query:

```
SELECT *
WHERE { ?m actor Tarantino . ?m budget ?b .
   ?m gross ?g }
```

Suppose we also have a statement asserting that DBpedia is complete for "the budget and gross of movies starring Tarantino", or in the abstract syntax:

```
Compl(?m budget ?b . ?m gross ?g | ?m actor Tarantino )
```

Note that by the condition, we can express that DBpedia has complete data about budget and box-office gross of movies starring Tarantino, even if in DBpedia Tarantino may not be listed as actor of all such movies. Now, none of the two sources alone is sufficient to answer this new query completely. Suppose as well that we have mappings using the RDFS predicates `subclass` and `subproperty` that associate terms in DBpedia to their LinkedMDB counterparts, if they exist, and vice versa. In this situation, CORNER can rewrite the original query in such a way, using SPARQL `SERVICE` calls [91], that each source contributes parts of a query for which they are complete. In our example, CORNER sends the subquery asking for movies starring Tarantino to LinkedMDB and the subquery asking for the budget and box-office gross to DBpedia:

```
SELECT *
WHERE {
   SERVICE <http://linkedmdb.org/sparql>
   { ?m actor Tarantino }
   SERVICE <http://dbpedia.org/sparql>
   { ?m budget ?b . ?m gross ?g } }
```

Practical requirements. In the beginning of this chapter, we introduced the completeness life cycle (as in Figure 7.1). We have also seen how CORNER could be used in the motivating scenario above. We now translate the above considerations into the following requirements, that CORNER should be able to:

1. Create and view completeness statements, as well as import and export completeness statements in RDF;
2. Set RDFS ontologies that may contain mappings between classes and properties of multiple data sources;
3. Check query completeness, and support federated rewriting where query parts are sent to the sources that can give complete results; and
4. Additionally as a demonstrator, switch on and off the completeness statements and RDFS mappings for giving ideas how completeness reasoning works.

7.1.2. System Architecture

Here we show how CORNER is built to satisfy the requirements above. As shown in Figure 7.2, CORNER consists of two main components, and is connected to the Linked Data layer (for query evaluation).

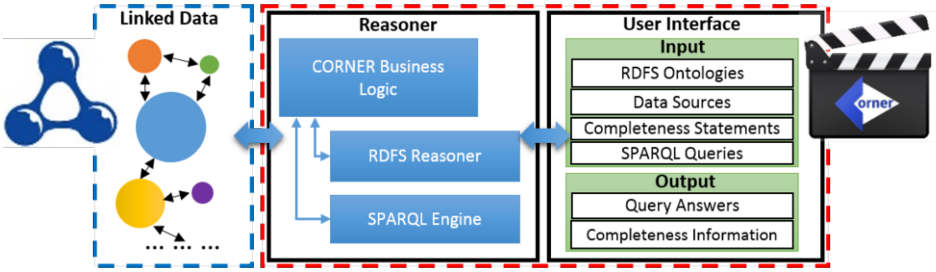


Figure 7.2. CORNER Architecture

The first component is the user interface (UI), which is developed using the Google Web Toolkit (GWT).[2] The UI provides users with the possibility to create and view completeness statements over data sources, as well as RDFS ontologies and queries. With the UI, it is also possible to switch on and off those elements, for instance, which completeness statements a user wants to consider in query completeness checking.

The second component is the reasoner, the backend of CORNER. The reasoner is implemented using Apache Jena.[3] The backend supports importing and exporting statements in RDF, and performs

[2]http://www.gwtproject.org/
[3]http://jena.apache.org/

data-agnostic completeness reasoning based on the inputs. The RDFS reasoner is needed since CORNER takes into account RDFS ontologies. If a query can be ensured to be complete, CORNER rewrites the query into a complete federated version and executes it over Linked Data. For this, the SPARQL engine is necessary. The query results along with the completeness information are given back to the users via the UI. The processes inside the backend are controlled by the CORNER business logic, which implements the data-agnostic completeness reasoning technique (as in Section 2.4), extended with the RDFS and federated features [27]. To take into account RDFS inferences, one needs to apply the RDFS closure computation [81] before and after the $T_{\mathbf{C}}$ operation, and check if the prototypical graph $\tilde{P}$ is included in the RDFS-enriched $T_{\mathbf{C}}$ application results. For the federated extension, basically the $T_{\mathbf{C}}$ operation needs to be modified such that it gives data source annotations to parts of the prototypical graph captured by the data source's statements. Then, those parts can be evaluated over the respective data sources according to the annotations using the `SERVICE` operator.

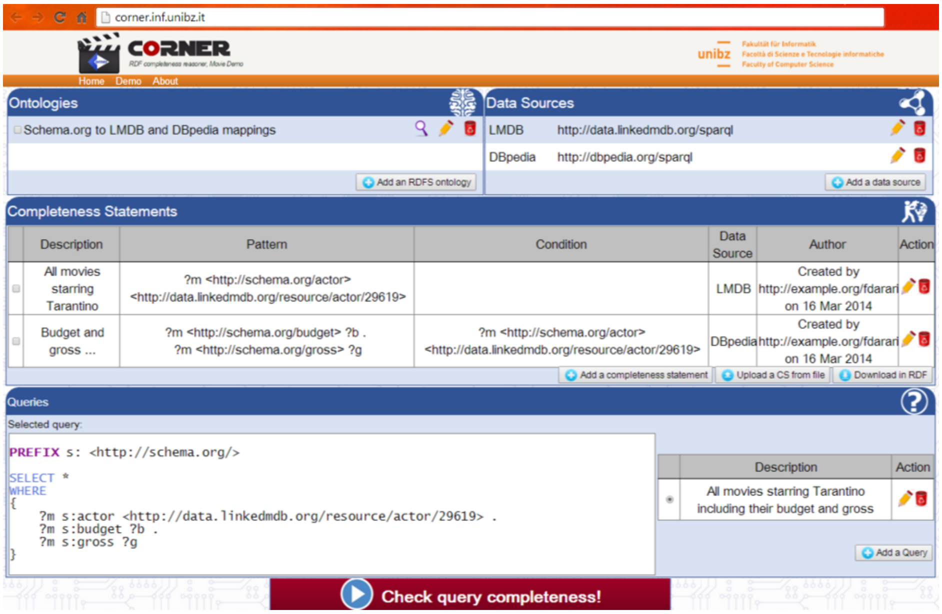
Reasoner implementation using Jena. We describe how Java with the Apache Jena library[4] can be used to develop the CORNER reasoner. As described in Section 2.4, the core of the data-agnostic completeness reasoning is the containment checking $\tilde{P} = T_{\mathbf{C}}(\tilde{P})$ where $\tilde{P}$ is the prototypical graph of a query $Q = (W, P)$ and $\mathbf{C}$ is a set of completeness statements. For completeness statements, we create the class `CompletenessStatement` with the field `pattern` and `condition`. Each completeness statement has an associated `CONSTRUCT` query, which can be realized in Java using the Jena class `Query`. The input query to be checked for completeness is also an instance of the class `Query`. Now for the reasoning, we create the method `freeze` to get the prototypical graph of the query, where we use the Jena class `Model` to realize the graph. From the resulting prototypical graph, we build the method `tcOperator` that evaluates all `CONSTRUCT` queries of the statements over the graph. Then, we decide query completeness by checking if the original prototypical graph is contained in the evaluation results. For the RDFS extension, we rely on Jena's `Reasoner` class which supports RDFS inferences, whereas the `SERVICE` operator is supported by the Jena class `QueryExecution`. Note that though our

[4]`http://jena.apache.org/`

implementation uses Apache Jena, any other off-the-shelf Semantic Web library like RDF4J (`http://rdf4j.org/`) can also be utilized.

7.1.3. UI Description

From the CORNER Web UI, to support the practical requirements, users may add RDFS ontologies, data sources, completeness statements of a specific data source, and queries, in addition to those already there. There is a panel in CORNER for each type of information. There are also the options to upload and download CORNER completeness statements in RDF in order to embed them into some existing metadata descriptions of data sources like `VoID`. When adding a new completeness statement, users see a pop-up window where they can specify patterns, conditions, the data source where the statement holds, the author and a description of the completeness statement. When checking the completeness of a query, CORNER displays a pop-up window comprising completeness information about the query, the query results, the debugging information, the ontologies used in the reasoning, a federated rewriting of the query, and the author information for each completeness statement.



Schema.org to LMDB and DBpedia mappings	

LMDB	http://data.linkedmdb.org/sparql
DBpedia	http://dbpedia.org/sparql

	Description	Pattern	Condition	Data Source	Author	Action
	All movies starring Tarantino	?m <http://schema.org/actor> <http://data.linkedmdb.org/resource/actor/29619>		LMDB	Created by http://example.org/fdarari on 16 Mar 2014	
	Budget and gross ...	?m <http://schema.org/budget> ?b . ?m <http://schema.org/gross> ?g	?m <http://schema.org/actor> <http://data.linkedmdb.org/resource/actor/29619>	DBpedia	Created by http://example.org/fdarari on 16 Mar 2014	

```
PREFIX s: <http://schema.org/>

SELECT *
WHERE
{
    ?m s:actor <http://data.linkedmdb.org/resource/actor/29619> .
    ?m s:budget ?b .
    ?m s:gross ?g
}
```

	Description	Action
●	All movies starring Tarantino including their budget and gross	

Figure 7.3. CORNER Homepage

Figure 7.3 shows the example of the query about budget and box-office gross of movies starring Quentin Tarantino, mentioned above. We first specify the SPARQL query in the query panel of the Web UI. Then, in the ontology panel, we specify which ontologies we want to use. In this case, we need to use the mapping ontology for LinkedMDB and DBpedia. After that, in the completeness statements panel, we select the statements about data sources to be used for query completeness checking. The figure shows the two completeness statements we mentioned above.

To start completeness reasoning, the user has to click the execution button at the bottom of the UI. Now, CORNER returns to the user the query results and information stating that the completeness of the query can be guaranteed. CORNER also provides debugging information about the completeness reasoning and the federated rewriting of the query that was executed over the data sources.

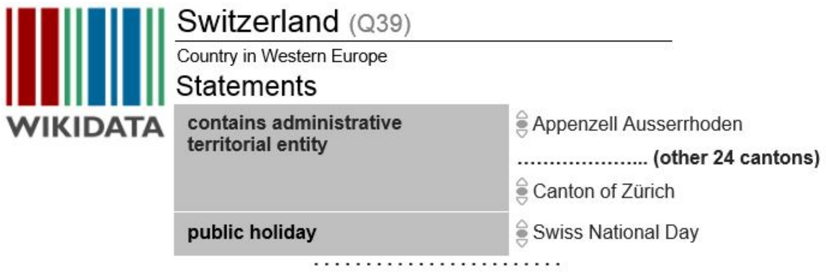
7.2. COOL-WD: A Completeness Demonstrator for Wikidata

In the previous section, we have seen how CORNER demonstrates a completeness hub to manage and consume completeness statements across multiple data sources. Now we explore how to build a system to support the management and consumption of completeness statements over a single data source. We chose Wikidata, which is entity-centric and crowdsourced, as a case study for our demonstration system, due to its recent popularity and relatively good quality. We first provide a motivating scenario and analyze practical requirements to build such a system. Then, we describe how SP-statements can be a suitable fragment of completeness statements for Wikidata. Next, we explore various sources of completeness statements that can be imported into our system. We provide a description of our system architecture, and then of how our system supports the consumption of completeness statements. Our demonstration system, called COOL-WD, is accessible at `http://cool-wd.inf.unibz.it/` and currently stores over 10,000 SP-statements.

7.2.1. Motivating Scenario

As a motivating scenario, let us consider the data about Switzerland (`https://www.wikidata.org/wiki/Q39`) on Wikidata as shown in

Figure 7.4. The page mentions the two properties "contains administrative territorial entity" and "public holiday".



Switzerland (Q39)

Country in Western Europe

Statements

contains administrative territorial entity	Appenzell Ausserrhoden
	.... (other 24 cantons)
	Canton of Zürich
public holiday	Swiss National Day

Figure 7.4. Wikidata page of Switzerland

At the moment of the writing, Wikidata contains 26 cantons of Switzerland from Appenzell Ausserrhoden to Canton of Zürich. According to the official page of the Swiss government,[5] there are exactly these 26 Swiss cantons and they are all stored in Wikidata. Therefore, as opposed to the public holidays, which are not complete,[6] the data about Swiss cantons in Wikidata is actually complete. However, Wikidata currently lacks support for expressing completeness information,. thus limiting the potential consumption of its data (e.g., assessing query completeness or performing completeness analytics). In general, not only Wikidata, but also other entity-centric KBs (e.g., DBpedia, YAGO) have such a limitation. We identify the following requirements of a completeness management system for Wikidata, which are in line with the completeness life cycle as in Figure 7.1. The system should be able to:

1. Create and update completeness statements, where the statements fit with the entity-centric, crowdsourced nature of Wikidata;
2. Make available the completeness statements in a machine-readable format;
3. View completeness statements together with the respective parts of data stated to be complete; and
4. Consume completeness statements such as by checking query completeness or performing completeness analytics.

[5]https://www.admin.ch/opc/de/classified-compilation/13.html

[6]There are at least 10 public holidays in Switzerland according to https://www.zuerich.com/en/visit/public-holidays.

7.2.2. SP-statements for Wikidata

Wikidata provides its information in an entity-centric way, that is, information is grouped into entities such that each entity has its own (data) page, showing the entity's property-value pairs. Furthermore, the data in Wikidata is curated by Wikidata users. In Section 3.3, we introduced SP-statements, that is, statements about the completeness of the set of values of a property of an entity. SP-statements are suitable for Wikidata due to its similarity wrt. the SPO-structure of Wikidata, and its simplicity. Having SP-statements over Wikidata provides structured, explicit information about the completeness of Wikidata.

SP-statements in RDF. To make SP-statements machine-readable, we want to make them available in practice by following the Linked Data principles:[7] SP-statements should be identifiable by URIs and accessible in RDF. To improve usability, URIs for SP-statements should indicate which entity is complete for what property. As an illustration, for our system we identify the SP-statement (*Switzerland*, *canton*)[8] with the URI `http://cool-wd.inf.unibz.it/resource/statement-Q39-P150`, indicating the entity Switzerland (Wikidata ID: Q39) and the property "contains administrative territorial entities" (Wikidata ID: P150).

Next, looking up an SP-statement's URI must give a description of the statement. Thus, we provide RDF modeling of SP-statements. We divide the modeling into two aspects: core and provenance. The core aspect concerns the intrinsic aspect of the statement, whereas the provenance aspect deals with the extrinsic aspect of the statement, providing information about the generation of the statement. The provenance aspect of SP-statements can also serve as a basis for trust determination over query completeness checking (e.g., "this query is complete based on the completeness assertions X, Y and Z, given by A and B on date D, with references to R and S"). The core aspect consists of the type of the resource, the subject and predicate of the SP-statement, and the dataset to which the statement is given (in the scope of this work, the dataset is Wikidata). This dataset information is particularly useful in, e.g., metadata integration. The provenance aspect consists of the author of the statement, the times-

[7] `https://www.w3.org/DesignIssues/LinkedData.html`

[8] For readability purposes, we represent SP-statements as pairs (S, P) instead of the full representation $Compl((S, P, ?v))$.

tamp when the statement is generated, and the primary reference of the statement. For the core aspect, we developed our own vocabulary, available at `http://completeness.inf.unibz.it/sp-vocab`. For the provenance aspect, to maximize interoperability we reused the W3C PROV ontology.[9] The following is RDF modeling of the SP-statement "Complete for all cantons in Switzerland" for Wikidata.[10]

```
wd:Q2013 spv:hasSPStatement coolwd:statement-Q39-P150 . # Q2013 = Wikidata
coolwd:statement-Q39-P150 a spv:SPStatement ;
  spv:subject wd:Q39 ; # Q39 = Switzerland
  spv:predicate wdt:P150 ; # P150 = canton
  prov:wasAttributedTo [ foaf:name "Fariz Darari" ;
    foaf:mbox <mailto:fariz.darari@stud-inf.unibz.it>] ;
  prov:generatedAtTime "2016-05-19T10:45:52"^^xsd:dateTime ;
  prov:hadPrimarySource
<https://www.admin.ch/opc/en/classified-compilation/19995395/index.html#a1>.
```

In the RDF snippet above, we see all the core and provenance aspects of the SP-statement for Wikidata of all cantons in Switzerland. The data source Wikidata is identified by `wd:Q2013`, having the SP-statement about the completeness of the cantons of Switzerland. The statement is of type `spv:SPStatement`, and has the `spv:subject` of Switzerland, and the `spv:predicate` of canton (or "contains administrative territorial entity"). The author attribution, timestamp, and reference use `prov:wasAttributedTo`, `prov:generatedAtTime`, and `prov:hadPrimarySource`, respectively.

Having such snippets would also provide the possibility to export SP-statements about a dataset into an RDF dump, which may then be useful for data quality auditing or completeness analytics purposes.

7.2.3. Creating Completeness Information

In general, one can imagine that SP-statements could either originate from (*i*) KB contributors, (*ii*) paid crowd workers, or (*iii*) web extraction. While our focus with COOL-WD is to provide a system for the first purpose, we used also other methods to pre-populate COOL-WD with completeness information.

KB Contributors. Wikidata already provides a limited form of completeness statements: no-value statements, as described in Section 3.4. SP-statements can be used to capture no-value statements, with the

[9] `http://www.w3.org/ns/prov`

[10] As always, the prefix declarations are provided in Appendix A.

condition that the respective data stated in SP-statements is empty in the data graph. We imported about 7600 no-value statements from Wikidata.[11] The top-three properties used in no-value statements are "member of political party" (12%), "taxon rank" (11%), and "follows" (11%). The properties "spouse", "country of citizenship", and "child" are among the top-15.

Paid Crowd Workers. Given the simplicity of SP-statements, it is natural to think of crowdsourcing as a way to generate SP-statements. We imported around 900 SP-statements from the crowdsourced statements in the work of Galárraga et al. [39], originally used for completeness rule mining. Regarding the crowdsourcing done in that work, it is noteworthy that it comes with issues. The first is the price, about 10 cents per statement. The other is, that crowd workers did not truly provide completeness assertions, instead, they were asked, whether they could find additional facts on a limited set of webpages. Truly asking crowd workers for checking for evidence for completeness was deemed too difficult in that work.

Web Extraction. From Mirza et al. [78], we imported about 2200 completeness assertions for the *child* relation, that were created via web extraction. These statements are all about the "child" property in Wikidata, and were generated as follows: 30 regular expressions were manually created and were used to extract information about the number of children from biographical articles in Wikipedia. For instance, the pattern "X has Y children" would match the phrase "Obama has two children and lives in Washington", and can thus be used to construct the assertion that Obama should have exactly two children. In total, they found about 124,000 matching phrases, of which, after filtering some low-quality information, about 84,000 phrases that had a precision of 94% were retained. For each of these 84,000 assertions, it was then checked whether the asserted cardinality matched the one found in Wikidata. If that is the case, it was then concluded that Wikidata is complete wrt. the children of the person. For instance, for Obama one truly finds two children in Wikidata, and thus, assuming the correctness of the phrase in Wikipedia, can conclude that Wikidata is complete. Later in Chapter 8, we provide a generalization over this work, by providing an automated method

[11] `https://www.wikidata.org/wiki/Help:Statements#Unknown_or_no_values`

of relation cardinality extraction from text.

7.2.4. COOL-WD Architecture

COOL-WD is a web-based completeness demonstrator for Wikidata, that provides a way to annotate complete parts of Wikidata in the form of SP-statements. COOL-WD focuses on the direct-statement fragment of Wikidata, in which neither references nor qualifiers are being used. A COOL-WD user is able to view data of Wikidata entities that has been annotated with SP-statements. A complete property of an entity is denoted by a green checkmark, while a possibly incomplete one is denoted by a gray question mark. A user can add a new SP-statement for a property of an entity by clicking on the gray question mark. In Subsection 7.2.5, we discuss several ways to consume completeness statements in COOL-WD.

In providing its features, COOL-WD maintains real time communication with Wikidata. On the client side, user action like entity search is serviced by MediaWiki API[12] calls towards Wikidata, while on the server side the COOL-WD engine retrieves entity and property information via SPARQL queries over the live Wikidata SPARQL endpoint.[13] User-provided SP-statements are stored in a specialized database. The engine retrieves SP-statements from the database to annotate the entities and properties obtained from the Wikidata SPARQL endpoint with completeness information, and then sends them to the user via a HTTP connection. The engine also manipulates the DB whenever a user adds a new SP-statement, and supports the data-aware completeness reasoning algorithm as described in Section 3.2.

Hardware and system specification. Our web server and database server run on separate machines. The web server is loaded with 16GB of virtual memory and 1 vCPU of 2.67GHz Intel Xeon X5650. It runs on Ubuntu 14 and uses Tomcat 7 and Java 7 for the web services. The database server has 8GB of virtual memory and 1 vCPU of 2.67GHz Intel Xeon CPU X5650, running on Ubuntu 12 and using PostgreSQL 9.1.

COOL-WD Gadget. Our external system of COOL-WD provides the

[12]`https://www.wikidata.org/w/api.php`
[13]`https://query.wikidata.org/bigdata/namespace/wdq/sparql`

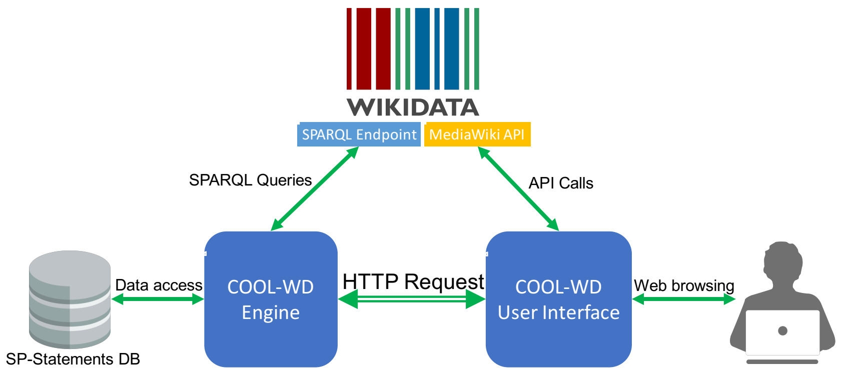


Figure 7.5. System architecture of COOL-WD

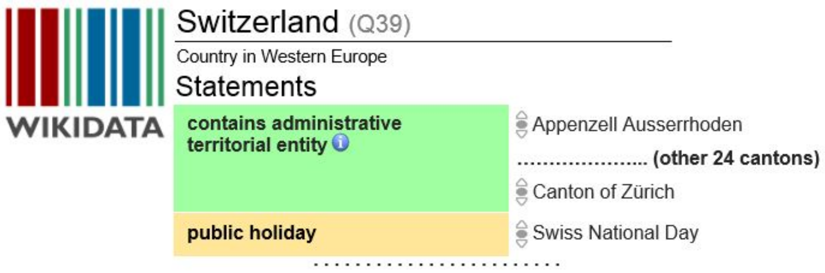

WIKIDATA
Switzerland (Q39)
Country in Western Europe
Statements
contains administrative territorial entity
Appenzell Ausserrhoden
.... (other 24 cantons)
Canton of Zürich
public holiday
Swiss National Day

Figure 7.6. COOL-WD Gadget: The green box indicates completeness, while the yellow box indicates potential incompleteness

full functionality of managing and consuming completeness information over Wikidata. Nevertheless, a Wikidata user might prefer to view and add completeness statements directly inside Wikidata. Here we present a Wikidata user script that enables such a functionality.[14] To activate the script, a user needs a Wikimedia account. Then, she has to import it to her common.js page at `https://www.wikidata.org/wiki/User:[wikidata_username]/common.js`. Basically, the script makes API requests to our own completeness server that provides a storage service for completeness statements.

Figure 7.6 shows that the property box of "contains administrative territorial entity" for Switzerland is colored green, which indicates completeness. The information icon "(**i**)", when clicked, provides the reference URL, the Wikidata username, and the timestamp of the

[14]`https://www.wikidata.org/wiki/User:Fadirra/coolwd.js`

completeness statement. Note that for properties not yet known to be complete, they will be colored yellow. To add a completeness statement one simply clicks the yellow property box. By clicking the information icon, one can add the reference URL of the statement in the provided pop-up form.

7.2.5. Consuming SP-Statements

The availability of completeness information in the form of SP-statements opens up novel ways to consume data on Wikidata, realized in COOL-WD.

Data Completion Tracking. The most straightforward impact of completeness statements is that creators and consumers of data become aware of its completeness. Thus, creators know where to focus their efforts, while consumers become aware of whether consumed data is complete, or whether they should take into account that data may be missing, and possibly should do their own verification, or contact other sources. Figure 7.7 illustrates the progress in completing information about Barack Obama in Wikidata via COOL-WD.

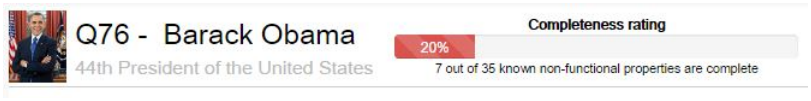

Q76 - Barack Obama
44th President of the United States
Completeness rating
20%
7 out of 35 known non-functional properties are complete

Figure 7.7. Via COOL-WD, we know that Wikidata is complete for 7 out of 35 known non-functional properties of Barack Obama (`http://cool-wd.inf.unibz.it/?p=Q76`)

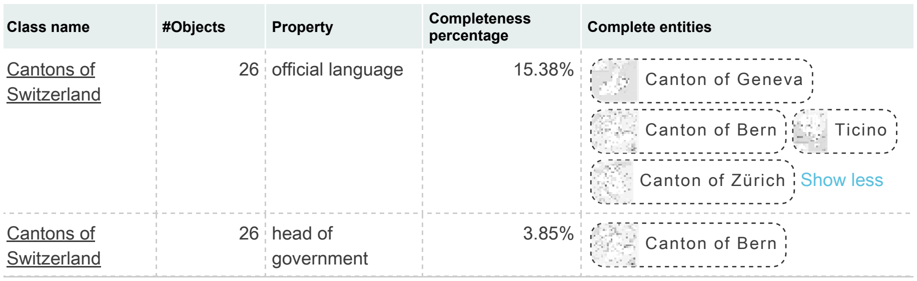
Completeness Analytics. Having completeness statements allows us to analyze how complete an entity is compared to other similar entities. For example, Wikidata might have complete information about the official languages of some cantons of Switzerland, but not all. A Wikidata contributor could exploit this kind of information to spot some entities that are less complete than other similar ones, then focus their effort on completing them.

Here we give a possible use case of completeness analytics. In COOL-WD, a class of similar entities is identified by a SPARQL query. For example, the class of all cantons of Switzerland consists of the entities returned by the query

```
SELECT * WHERE { wd:Q39 wdt:P150 ?c }
```

where Q39 is the Wikidata entity of Switzerland and P150 is the Wikidata property "contains administrative territorial entity." A user may add a new class by specifying a valid SPARQL query for the class. Then, COOL-WD would list all possible properties of the class by taking the union of the properties of each entity of the class. The user would then be asked to pick some properties that they feel important for the class.

Now, suppose we pick "official language" and "head of government" as important properties for the cantons of Switzerland, and suppose that we have only the following SP-statements: *(Bern, lang), (Geneva, lang), (Ticino, lang), (Zurich, lang), (Bern, headOfGov).* Figure 7.8 shows how COOL-WD displays such analytics information. A user then can see that Wikidata is verified to have complete information about official languages only for 4 out of 26 cantons of Switzerland (15.38 %), which means that the remaining 22 cantons are possibly less complete than the four. Wikidata has also complete information for the head of government of Canton of Bern, only one out of the 26 cantons. Using this information, a contributor is able to focus on checking and completing the languages of the remaining 22 cantons and the head of government of the other 25 cantons.



Class name	#Objects	Property	Completeness percentage	Complete entities
Cantons of Switzerland	26	official language	15.38%	Canton of Geneva, Canton of Bern, Ticino, Canton of Zürich, Show less
Cantons of Switzerland	26	head of government	3.85%	Canton of Bern

Figure 7.8. An overview of the completeness analytics feature. Clicking on the class name shows a more detailed analytics of the class.

Query Completeness Assessment. With explicit completeness information over data comes the possibility to assess query completeness. In COOL-WD, it is possible to perform data-aware completeness checking, as discussed in Chapter 3. Here obviously the data graph is Wikidata. We further extend the query checking feature with a diagnostics feature: depending on whether a query can be guaranteed to be complete or not, users may also see either all SP-statements, including their provenance (i.e., author, timestamp, and reference URL), contributing to the query completeness, or a missing SP-statement as a cause of no completeness-guarantee. Wrt. our data-aware reasoning algorithm as in Section 3.2, the finding of such a cause is by taking a saturated part that is not contained in the data graph.

Let us give an illustration on how query completeness diagnostics works. Consider the query "give all languages of all cantons in Switzerland." Suppose we have the following SP-statements:

$$(Switzerland, canton), (Aargau, lang), \ldots, (Zurich, lang).$$

The statements ensure the completeness of all cantons in Switzerland, and for each canton of Switzerland from Aargau to Zurich, the completeness of all languages. The diagnostics feature enables that: in addition to the information that the query is complete, users see all those SP-statements (and their provenance) that contribute to the query completeness. In contrast, suppose that we do not have the SP-statement $(Zurich, lang)$. In this case, the query cannot be guaranteed to be complete, and the diagnostics feature reports that $(Zurich, lang)$ is missing.

7.3. Related Work

One of the systems related to our work is MAGIK [101], which allows one to collect completeness information about relational databases, and to use it in query answering. The system was based on the work in [96] about completeness reasoning over relational databases. MAGIK implemented the reasoning by translating completeness reasoning tasks into logic programs, which are evaluated using an answer set engine. Our work focuses more on Semantic Web data and queries, as opposed to MAGIK. Chu et al. [20] developed KATARA, a hybrid data cleaning system, which not only cleans data, but may also add new facts to increase the completeness of the KB. KATARA performed data cleaning by establishing some correspondence between the possible dirty database with the available knowledge bases (KBs), and leveraging human involvement for data verification when the KBs lack coverage. Acosta et al. [4] developed HARE, a hybrid SPARQL engine to enhance answer completeness. HARE implemented query execution techniques that can identify portions of queries that yield missing values. Then, in order to resolve missing values, HARE performed microtask crowdsourcing. As opposed to our work, KATARA and HARE cannot be used to check whether queries are complete in the sense that *all* answers are returned, as they focus more on increasing the degree of KB and query completeness.

7.4. Summary

In this chapter, we have identified practical requirements and demonstrated how systems to support the completeness life cycle of RDF data sources can be built. The first demonstration system is CORNER. CORNER serves as a hub of completeness statements over multiple data sources. CORNER supports data-agnostic completeness reasoning, with the RDFS inference and federated rewriting features. The second demonstrator is COOL-WD. COOL-WD showcases a completeness management functionality over Wikidata. With COOL-WD, users can add and view SP-statements of Wikidata entities. SP-statements in COOL-WD are available also via a Linked Data API. Moreover, the COOL-WD consumption functionalities consist of data completion tracking, completeness analytics, and data-aware query completeness assessment. For flexibility, COOL-WD comes in two variants: as an external system, or as a gadget where edits of completeness statements can be performed directly over Wikidata.

An open, practical issue is the semantics of completeness for less well-defined predicates such as "medical condition" or "educated at," as detailed in [98]. When it is unclear what counts as a fact for a predicate, it is also not obvious how to assert completeness. A possible solution is to devise a consensus or guidelines on what it means by a (complete) property, for instance: IMDb guidelines on complete cast or crew at `https://contribute.imdb.com/updates/guide/complete`. Further, the subjectivity of completeness along with potential impacts of COOL-WD has been discussed with the Wikidata community.[15,16]

[15] `https://lists.wikimedia.org/pipermail/wikidata/2016-March/008319.html`

[16] `https://lists.wikimedia.org/pipermail/wikidata/2016-August/009388.html`

Chapter 8

Extracting Relation Cardinalities from Text

In the previous chapter, we have demonstrated how completeness statements can be created manually. To improve the scalability, an automated method of generating completeness statements is thus important. Meanwhile, the Web contains a wealth of information about relation cardinalities, as exemplified by the sentence "Trump has five children" on Trump's Wikipedia page.[1] Intuitively, such information gives a hint on the complete count of the respective relation, which can be leveraged to assess the completeness of a knowledge base (KB) in the following way: Whenever the cardinality information matches the number of relation values of the entity in the KB, then this indicates that the KB is complete for that relation of the entity. Hence, a completeness statement can be generated.

Motivated by this rationale, in this chapter we introduce the novel problem of extracting cardinalities from text on the Web, and develop a CRF-based method for the problem. We employ distant supervision using fact counts in the KB as training data, encountering incompleteness as a new challenge wrt. classical fact extraction. We analyze linguistic particularities of cardinality information, and show that our method can achieve between 38% and 84% of precision on four human-evaluated relations. We also analyze the presence of cardinality information for more than 200 relations in Wikidata.

Preliminary results of this chapter have been published in [76], while the full results have been published in [77].

[1] `https://en.wikipedia.org/wiki/Donald_Trump` (as of May 29, 2017)

8.1. Introduction

General-purpose RDF knowledge bases such as Wikidata [111], DBpedia [10], or YAGO [108] find increasing use in applications such as question answering, structured search, or document enrichment, and their automated construction from text has received considerable attention. So far, construction techniques are focused on the extraction of fully qualified facts, but more often than not texts only contain relation cardinality information, i.e., the number of objects that stand in a relation with a certain subject, such as *"John has two children"* or *"Mary wrote 5 books"*, without mentioning the actual objects.

Extracting such relation cardinality information can hugely extend the scope of knowledge bases, thus allowing more accurate answers for queries that involve counts or existential quantification. For the child relation, for instance, simple manual patterns could reveal the existence of 178% more children from Wikipedia, than are currently contained in Wikidata [78].

Another important use of relation cardinalities is KB curation [86, 115]. KBs are notoriously incomplete, contain erroneous triples, and are limited in keeping up with the pace of real-world changes. For instance, even for a person of importance like U.S. president James A. Garfield, while the Wikipedia text mentions 7 children, Wikidata contains only 4. Similarly, DBpedia contains an erroneous child of Judy Moran called "Moran_family", leading to a total children count of 3, while all other sources speak only of 2 children. Extracting the cardinalities of relations could help addressing both issues.

Extracting relation cardinalities is more difficult than classical fact extraction for several reasons. For instance, one can observe that cardinality information can be compositional, as in the following sentences:

> *"Trump has three children with Ivana, a daughter with Marla, and a 10-year-old son with his current wife, Melania."*

Here, the total children count of 5, is split across three different predicates: *children*, *daughter*, and *son*.

Another challenge lies in the training data. Relation extraction usually relies on distant supervision, i.e., uses facts already contained in a KB as positive examples for identifying further patterns. In the case of relation cardinalities, however, knowledge bases frequently contain counts that are lower than what is correct.

Relation cardinalities are not extracted by state-of-the-art information extraction systems. ClausIE [34], for example extracts from the sentence *"Donald Trump has five children"* the triple ⟨*DonaldTrump, has, fiveChildren*⟩, i.e., it fails to recognize that *'five'* should be treated as parameter, not as part of the predicate. While IE methods that hinge on pre-specified relations for KB population (e.g., NELL [79]) can already capture numeric values for a few attributes such as ⟨*Berlin2016attack, hasNumOfVictims, 32*⟩, they are currently not able to learn them.

8.2. Relation Cardinalities

We define a mention of *relation cardinality* as follows:

> *"A cardinal number or a number-related term that characterizes the cardinality of a set of objects that stand in a specific relation with a certain subject."*

For example, in *"Mary has* **one** *son and identical* **twin** *daughters,"* *'one'* and *'twin'* are the expressions we try to identify to determine the *hasChild* cardinality for Mary, which is 3.

Our analysis on random numbers from Wikipedia articles revealed that around 19% numbers express relation cardinalities, most frequently for topics such as *sport* (e.g., matches played, goals scored), *creative work* (e.g., books written, seasons in an episode), *organization* (e.g., number of members) and *family relations.* At present, tools such as the Stanford Named Entity (NE) tagger [70] only label such numbers unspecifically as NUMBER. Identifying which relations these expressions quantify would give them semantics.

Given the substantial occurrences of relation cardinalities, one may also wonder whether cardinality extraction can improve the existential coverage of KBs, i.e., the number of facts known to exist. To answer this question, we analyzed Wikipedia articles of 200 random persons, comparing the amount of existential information for the *hasChild* relation that can be retrieved by the following three methods: *(i) cardinality extraction*, where we focus on the relation cardinalities in the article; *(ii) counting names*, where we focus on the names of the children in the article; *(iii)* and *Wikidata triples*, where we count the children facts from the respective Wikidata pages. Note that the second method above corresponds to what standard relation extraction aims to achieve.

Source	subjects	objects
Wikipedia articles		
cardinality information	.120	.350
names	.070	.175
Wikidata triples	.025	.030

Table 8.1. Fraction of persons (n=200) whose Wikipedia articles contain children cardinality information, children names, or who have children on Wikidata, and number of children per each method

As shown in Table 8.1, cardinality information allows to find children counts for 12% of all people, while names are only mentioned for 7%, and Wikidata contains children for only 2.5%. Similarly, with respect to the number of children in total, cardinality information allows learning of the existence of twice as many children as information extraction, and eleven times as many children as Wikidata knows of.

We conjecture that cardinality information can benefit both standard relation extraction, i.e., reducing false positives by extracting facts with high confidence only until a certain number of facts is reached, and question answering, as many questions such as *"Which US presidents were married thrice?"* only require knowledge of counts.

8.3. Relation Cardinality Extraction

Problem Statement. Given a relation/predicate p, a subject s and a corresponding text about s, we aim to extract the *relation cardinality*, i.e., the count of $\langle s, p, * \rangle$ triples, from relation cardinality mentions in the text.

Methodology. We approach the problem via sequence labeling, i.e., given a sentence containing at least one number, we employ a classifier to determine for each number in the sentence whether it is a mention of the cardinality of the relation of interest. We use CRF++ [62] to build a Conditional Random Field (CRF) based classification model for each relation, taking as features the context lemmas (window size of 5) around the observed token t, along with bigrams and trigrams containing t. Note that we use `_num_` as the lemma of each cardinal number found in the text, and multi-word numbers such as *'twenty one'* are collapsed into single tokens.

The relation cardinality of a given $\langle s, p \rangle$ pair is predicted by selecting the number in the text positively annotated by the classifier, which has marginal probability–resulting from forward-backward inference–higher than 0.1. If there are several such numbers in the text, the one having the highest probability is chosen.

Distant Supervision. We rely on distant supervision to generate training data. Given a knowledge base predicate p, for each entity s that appears as subject of p, we retrieve the triple count $\langle s, p, * \rangle$ from the knowledge base and a text about s. In particular, we use Wikidata as knowledge base and the Wikipedia page of each entity as text source, both in their version as of March 20, 2017.

We generate training data by annotating *candidate numbers*[2] in the text as correct cardinalities whenever *(i)* they correspond to the exact triple count and *(ii)* if they modify a noun,[3] i.e., there is an incoming dependency relation of label *nummod* according to the Stanford Dependency Parser [70]. Otherwise, they are labelled as O (for Others), like the rest of non-number tokens.

Dataset. We chose four Wikidata predicates that span various domains: *child* (P40), *spouse* (P26), *has part* (P527) and *contains administrative territorial entity* (P150)–or simply, *contains admin.* While the subjects of *contains admin*, *child* and *spouse* relations are of fairly uniform type (mostly *administrative territorial entity* and *human*), the *has part* relation is used in highly diverse domains, ranging from chemical substances and groups of buildings to organizations. We focused on two classes of subjects for *has part*: *series of creative works* (e.g., film series, novel) and *musical ensemble* (e.g., band).

Considering only subjects of the abovementioned predicates that have links to English Wikipedia pages, we set aside 200 random subjects for each predicate as *test set*; 100 instances of each class for *has part* relation. The remaining subjects that have at least one $\langle s, p, * \rangle$ triple are used as *training set.* Furthermore, we set aside 200 random subjects per predicate from the training set as *validation set.* Table 8.2 reports the number of subjects (#s) for each considered predicate (p) in the training set.

[2]Numbers that are not labelled as DATE, TIME, DURATION, SET, MONEY and PERCENT by the Stanford NE-tagger.

[3]This is to exclude numbers as in "*one* of the reasons..." from positive training examples.

p	$\#s$
has part	
- series of creative works	614
- musical ensemble	8,750
contains admin	6,118
child	38,496
spouse	43,668

Table 8.2. Number of Wikidata instances as subjects ($\#s$) of each predicate (p) in the training set

Evaluation. We report in the first rows of Table 8.3, the performance of our CRF-based method (vanilla) in predicting relation cardinalities, evaluated on the validation set. While we initially wanted to use knowledge base counts for the evaluation, it turned out that these were too often too low, thus we manually annotated the validation set with the true relation counts. Moreover, whenever the predicted number and the relation count matches, we manually check whether the textual evidence, i.e., sentence containing the predicted number, truly expresses the relation of interest.

We initially built one classifier for each predicate. However, we noticed that if we use distinct classifiers for each class in *has part*, i.e. one for *creative works* and another for *musical ensemble*, the performance improved considerably, particularly for *creative works* (.222 vs .372 F1-score). The method works reasonably well for *creative works* and *contains admin*, with .372 and .325 F1-scores, respectively. For *musical ensemble* and *spouse*, on the other hand, both precision and recall suffer, resulting in an overall performance of only around 2% F1-score.

We next discuss major limitations of the vanilla approach as revealed by the qualitative evaluation, and how to tackle them.

8.4. Improving Relation Cardinality Extraction

8.4.1. Training Data Quality

Unlike training data for normal fact extraction, which is generally highly correct (e.g., YAGO claims 95% precision [108]), taking triple counts found in knowledge bases as ground truth generally gives wrong results. For example, our manual annotation of the validation

	has part						contains admin			child			spouse		
	creative works			musical ensemble											
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
combined	*.238*	*.208*	*.222*	*.030*	*.023*	*.026*									
vanilla	**.421**	**.333**	**.372**	**.016**	**.011**	**.013**	**.660**	**.216**	**.325**	**.200**	**.159**	**.177**	**.028**	**.017**	**.021**
Training Data Quality															
ignore $n > c$	-.04	0	-.02	+.02	**+.02**	**+.02**	-.09	+.01	-.01	+.01	+.03	+.02	-.00	+.01	+.00
$c < n \leq c+1$	-.01	0	-.00	-.00	0	0	0	0	0	+.01	+.01	+.01	-.01	-.01	-.01
$c < n \leq c+2$	+.00	**+.01**	**+.01**	+.01	+.01	+.01	0	0	0	+.02	+.02	+.02	+.00	**+.01**	**+.01**
$c < n \leq c+3$	-.00	+.01	+.01	+.01	+.01	+.01	0	0	0	+.03	**+.03**	**+.03**	-.01	0	-.00
exclude freq. n	**+.07**	-.02	**+.01**	**+.03**	+.01	**+.02**	**+.04**	-.01	-.00	**+.10**	+.06	**+.08**	-.03	-.02	-.02
$n \leq 1$	+.01	+.01	+.01	+.44	+.05	+.09	+.03	0	+.00	**+.07**	+.04	**+.05**	+.03	-.01	-.00
$n \leq 2$	**+.06**	+.02	**+.04**	**+.70**	+.05	**+.09**	+.14	-.01	+.01	+.16	-.07	-.03	**+.97**	0	**+.01**
$n \leq 3$	+.02	-.09	-.06	+.58	+.02	+.05	**+.16**	-.01	**+.01**	+.60	-.14	-.13	-.03	-.02	-.02
top 25%	0	0	0	0	0	0	0	0	0	-.01	-.01	-.01	-.00	0	-.00
50%	**+.01**	0	**+.00**	-.00	0	-.00	0	0	0	-.01	-.01	-.01	-.01	-.01	-.01
75%	0	0	0	-.00	0	-.00	0	0	0	-.00	-.01	-.01	-.00	0	-.00
best train	**.525**	**.323**	**.400**	**.714**	**.056**	**.104**	**.800**	**.209**	**.332**	**.377**	**.278**	**.320**	**1.00**	**.046**	**.087**
Compositionality															
comp	-.06	+.01	-.01	0	0	0	**+.06**	**+.18**	**+.20**	**+.01**	**+.01**	**+.01**	-.33	0	-.00
Linguistic Variance															
transform	**+.06**	**+.13**	**+.11**	**+.09**	**+.03**	**+.05**	0	0	0	-.01	-.01	-.01	-.15	**+.02**	**+.03**
transform 'a'	-.12	-.02	-.06	-.26	-.01	-.01	-.11	-.04	-.06	-.12	-.06	-.08	-.67	-.01	-.02
best final	**.587**	**.449**	**.509**	**.800**	**.087**	**.157**	**.855**	**.386**	**.532**	**.384**	**.290**	**.330**	**.846**	**.063**	**.116**

Table 8.3. Evaluation results on the validation set

set for *child* shows that about 50% of the KB counts are incorrect wrt. the knowledge one can derive from Wikipedia texts.

In [76], we showed that manually generated training data can hugely boost performance, however, obtaining sufficient quantities of manually annotated data is generally costly. We see several avenues to tackle the training data quality issue.

Incompleteness-resilient Distant Supervision. Triple counts in the knowledge base are often lower than what is correct, but rarely too high. During the training data generation, these incorrect counts will generate spurious negative examples. For example, recalling President Garfield, for whom Wikidata knows only 4 out of his 7 children, the number "seven" in the sentence *"In 1858, he married Lucretia; they would have* ***seven*** *children..."* on his Wikipedia page[4] would be labelled as negative example, leading to a lower probability for numbers appearing in similar contexts to be labelled as correct cardinalities.

Since there is no way to know whether higher numbers in the text are actually positive examples, one possible approach is to treat them

[4]`https://en.wikipedia.org/wiki/James_A._Garfield`

as neither positive nor negative examples, but simply remove them from the training set. We test two variations of this approach:

- Ignore $n > c$, i.e., we remove sentences that only contain numbers (n) that are higher than the triple count (c).
- Ignore $c < n \leq c+d$, i.e., we remove sentences that only contain numbers slightly higher than the triple count, for values of d between 1 and 3.

Excluding Uninformative Numbers. The more frequent a certain number occurs in a text, the more probable it is to occur in various contexts. As a way to give the classifier less noisy training examples, one might wish to filter out frequently occurring numbers irrespective of whether they match the triple count or not. Specifically, we experiment with labeling numbers that occur more than 5 times in a text as negative examples.

By Benford's law, lower numbers are more frequent than higher numbers. As a very simple heuristic, we thus also experiment with excluding all n, $1 \leq n \leq 3$ from the training examples.

Filtering Ground Truth. Instead of taking the triple counts for all subjects of a predicate as ground truth, one might trade size for quality. We rank the subjects according to their *popularity*, i.e., the number of triples/facts about them stored in the knowledge graph. We then experiment with using only the 25%, 50% and 75% most popular subjects as training data.

8.4.2. Compositionality

We observed that cardinalities for *contains admin* were often mentioned as a composition of several numbers, e.g., *"The Qidong county has **4** subdistricts, **17** towns and **3** townships under its juridiction."* This phenomenon is also observed for *child*, as exemplified at the beginning of Section 8.2.

In this work, we focus on number compositionality when a sequence of numbers occurs in the same sentence. In training data generation, if the sum of such a number sequence is equal to the triple count, we label all numbers in the sequence as positive examples.

In the prediction step, we predict the relation cardinality by summing up consecutive numbers labeled as positive with sufficient probabilities by the classifier. To avoid predicting the wrong cardinality in *"He had **four** children: **two** sons and **two** daughters"* we check the number sequence as follows: for a predicted number p labeled as positive, if the sum of the following numbers, that are also labeled as positives, is equal to p, we simply choose p as the correct relation cardinality. In the previous example, our method will predict *four* as the children count instead of *eight*.

8.4.3. Linguistic Variance

Our initial motivation was to make sense of the so far ignored large fraction of numbers that express relation cardinalities. However, we noticed quickly that relation cardinalities are frequently also expressed with other concepts related to numbers such as *trilogy* or *duo*.

We used the *relatedTo* relation in ConceptNet [106] for collecting terms related to numbers. We split the terms into two groups, those having Latin/Greek prefixes[5] and those not having them. For the first group, we generated a list of Latin/Greek prefixes, e.g., *tri-*, *quart-*, and a list of possible suffixes, e.g., *-logy*, *-et*. We manually checked the latter group to select only terms that were strongly associated with cardinalities, e.g., *twin*, *thrice* and *dozen*.

In a pre-processing step, a Latin/Greek number found in the text is represented with only its suffix as the lemma, and labelled as a positive example if its prefix corresponds to the relation count. For example, when we found *'triplet'* in the text, its lemma will be converted to `_plet_` and it will be labelled as a positive example if the relation count is equal to 3. For other terms, we simply replace them with the correct terms containing cardinal numbers, e.g., *twin* → *two children*, *thrice* → *three times* and *dozen* → *twelve*.

We also observed that the relation cardinality of *one* is frequently represented with indefinite articles, for instance, *"They had **a** son together"* or *"It has **a** residential community and 7 villages under its adminstration."* Therefore, we also experiment with converting indefinite articles *a* and *an* in the test/validation set into *one*.

[5]`http://phrontistery.info/numbers.html`

8.5. Analysis

8.5.1. Evaluation on the Validation Set

We performed an ablation study to identify the impact of each idea from above wrt. the vanilla approach. The results are reported in Table 8.3, based on the same evaluation methodology used in Section 8.3.

Training Data Quality. Ignoring numbers larger than KB counts was found to slightly improve the performance, except for *contains admin*. We presume the reason for this is that Wikidata is already highly complete for this relation. For other relations, the varying degree of deviation d that improves the performances hints at how many $\langle s, p, * \rangle$ triples per subject s are usually missing from the knowledge graph, i.e., $d = 3$ for *child*, and $d = 2$ for *creative works* and *spouse*. For *musical ensemble*, ignoring all higher numbers is the best approach, which suggests that Wikidata is remarkably incomplete for that relation.

Excluding numbers frequently occurring in the text turns out to considerably improve precision (except for *spouse*), for instance by 10% for *child*. Excluding low numbers has a similar effect, although the effect appears very much dependent on the nature of the predicates, i.e., the average number of $\langle s, p, * \rangle$ triples that are often mentioned as cardinality assertions for the observed predicate p in the text about s. For instance, when excluding $n \leq 1$ is the best setting for *child*, then that means that two children are frequently mentioned in texts, hence, excluding $n \leq 2$ would filter more positive than negative examples.

Somewhat surprisingly, taking smaller but more complete subsets for training did not have any effect on performance. We conjecture that for these instances, a more complete knowledge base is offset by longer and thus more noisy articles.

In Table 8.3, we report the extraction performance after our attempts to improve the training data quality (best train) by using the corresponding best setting (shown in bold) for each predicate. The *best train* scores are then used to further show the impact of tackling compositionality and linguistic variance discussed below.

Compositionality and Linguistic Variance. The results on tackling the compositionality and linguistic variance issues shed further light on the nature of each relation. Cardinality assertions for *contains admin* are very often compositional, as shown by the improvement of 20% in F1-score, seldom for *child* with 1% F1-score increase, and not at all for the others.

Instead, the other relations benefited from considering concepts related to numbers as candidates for relation cardinality. We observe significant improvements of both precision and recall for *has part*, and of recall for *spouse*. This approach allows the extraction method to infer the relation count from terms such as *'pentalogy'*, *'duo'* and *'(married) twice'*.

Transforming all indefinite articles *'a'* and *'an'* into *'one'* in the test data, in turn, results in a great increase of false positives, and reduces precision considerably.

The final performance of our extraction method for each relation on the validation set is shown in the last row (best final) of Table 8.3. The method works quite well for *contains admin*, *spouse* and *musical ensemble* with 85.5%, 84.6% and 80% precision scores respectively. The low recall for *musical ensemble* and *spouse* reflects the rarity of cardinality assertions containing cardinal numbers (or number-related terms) for those relations. Average performance with 50.9% F1-score on *has part* for *creative works* might be due to the comparably small training data set. Meanwhile, we attribute an observed lower precision on *child* to three factors:

1. The classifier often confuses the number of children with, for instance, number of siblings, spouses, or (political) terms served.
2. The number-of-children assertions found in the text (about a person) are actually about someone else, e.g., his/her parent or sibling.
3. The total number of children can be inferred from numbers mentioned in several sentences, as in *"John married Jane in 1983. They have **two** children together. After their divorce in 1995, he married Jamie, with whom he has **two** sons and **one** daughter."*

p	RCE			%subject			existential knowledge increase
	P	R	F1	RCE	Wikidata	Wikipedia	(Wikidata+RCE) / Wikidata
has part							
- creative works	.545	.279	.369	.120	.020	.550	17.3
- musical ensemble	.400	.026	.049	.020	.280	.770	1.1
contains admin	.571	.308	.400	.020	.060	.065	1.8
child	.625	.750	.682	.070	.020	.095	7.6
spouse	.500	.026	.050	.005	.020	.019	1.8

Table 8.4. Evaluation results on the test set; RCE denotes *Relation Cardinality Extraction.*

8.5.2. Evaluation on the Test Set

We also evaluated the performance of our method on the test data, which contains crowd-annotated 200 random entities per relation. We used the CrowdFlower[6] platform for annotating *(i)* whether the number of objects could be inferred from the Wikipedia page of a certain subject, and *(ii)* what that number was, taking in each case the majority vote among three crowdworkers. Quality was ensured via unambiguous test questions. It turns out that the task was not trivial, as on the random entities, annotators voted unanimously in only 83% of cases. Frequent reasons for disagreement were for instance for *has part*, when different granularities like *"3 seasons and 12 episodes"* were mentioned, or when for a band, a vocalist, two guitarists and a drummer were mentioned, but it was left unclear whether these were all members.

In Table 8.4, we report the performance of our method on the crowd-annotated dataset. The recall (RCE, R) was computed by using the total number of subjects of which the crowd could infer their object cardinality from Wikipedia articles. Our method could extract cardinality information with precision (RCE, P) ranging from 40% to 62.5%.

We also report in the next columns the percentage of subjects (%subject) for which *(i)* our method could extract the relation counts correctly (RCE), *(ii)* Wikidata contains at least one fact in the respective relation, and *(iii)* the crowd workers said one could infer the relation count by any means from the Wikipedia article. As one can see, for *contains admin* and *child* the percentage of subjects of which our method succeed in extracting the cardinalities is reasonably close to the ones of Wikipedia. For *creative works*, *musical ensemble* and

[6] https://www.crowdflower.com/

spouse, the large gap stems from the facts that Wikipedia articles more often mention the individual objects, which allows crowd workers to infer the cardinality by counting, a technique that is currently not accessible by our method.

In the existential knowledge increase column we report the impact of relation cardinality extraction towards enlarging the existential knowledge of KBs, in this case Wikidata. For *creative works* and *child*, the number of facts known to exist increased significantly, by 17.3 and 7.6 times respectively. Meanwhile, for *musical ensemble*, Wikidata usually already contains the ensemble member names, so extracting cardinality information does not help much.

8.6. Large-scale Run of Relation Cardinality Extraction

We collected all Wikidata properties that were not asserted to be single-value[7], had a functionality degree ($\#subjects/\#triples$) of less than 0.98 [40], and were used by at least 500 subjects, obtaining 267 properties in total.

For each property/relation, we set aside the 200 of the 400 most popular entities as test set, while using the rest (limited to 10k most popular entities) as training data. Note that we only considered entities of the most frequent type for each class, e.g., *human* for *sibling*, to ensure domain homogeneity. We then ran our Relation Cardinality Extraction (RCE) system for each property, using the setting we assume to generally work well for all relations (vanilla + ignore $c < n \leq c + 2$ + exclude freq. n + exclude $n \leq 1$). We evaluated the precision wrt. the triple counts for the entities in the test set, assuming that for the most popular entities, these are usually correct.

There were a total of 147 for which RCE could identify relation cardinalities with more than 5% precision. While some are spurious results due to low variance, in Figure 8.1 we show some properties where the results were manually found to be not mere coincidences. These properties are used, for instance, for humans (e.g., *sibling*, *award received*), games/software (e.g., *designed by*, *software version*), companies (e.g., *founded by*, *subsidiary*) and transportation-related

[7]Properties having the property constraint type https://www.wikidata.org/wiki/Q19474404

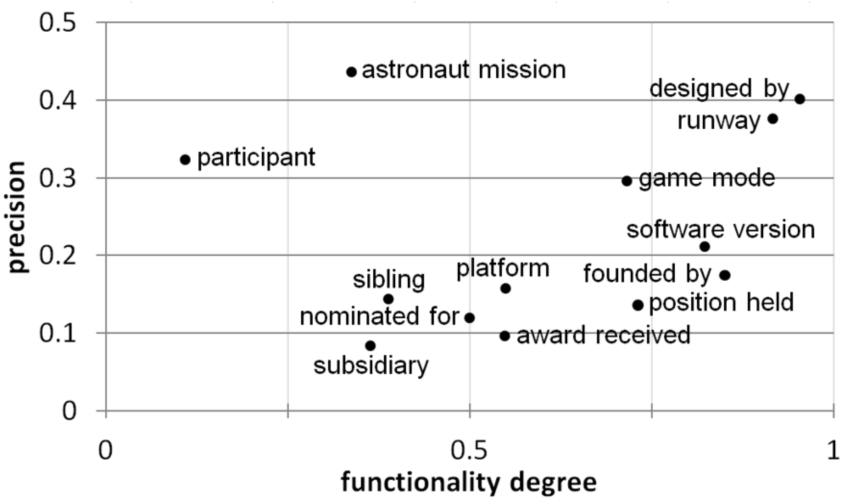


Figure 8.1. Precision results on some notable Wikidata relations, along with their corresponding functionality degrees

buildings (e.g., *platform*, *runway*). Our method also achieves an impressively high precision of 97.8% on *contains settlement*, which is a relation similar to *contains admin*.

8.7. Related Work

Advances on the automated construction of large-scale KBs have been largely influenced by prevalent relation extraction works, focusing either on structured data [108, 10] or on unstructured contents over the web. For the latter, directions include extracting arbitrary facts without predefined schema, called Open IE [72, 34, 79], and extracting triples based on well-defined knowledge base relations [109, 61, 85], in which the distant supervision approach is widely used [24, 75]. There has also been work on reducing noise in distantly-supervised training data via learning only from positive examples [74] or by expanding the knowledge base with information retrieval techniques [113].

Most relation extraction works have focused on non-numeric information. [69] explored relation extraction where one of the arguments is a number or a quantity (e.g., ⟨*Aluminium, atomicNumber, 13*⟩). In general, most works on making sense of numbers in texts or semi-structured data (e.g., web tables) have been largely focused on temporal information [65, 107] and physical quantities or measures [18, 57, 82].

In contrast, numbers that express relation cardinalities have received little attention so far. State-of-the-art Open-IE systems either hardly extract cardinality information or fail to extract cardinalities at all. While NELL, for instance, knows 13 relations about the number of casualties and injuries in disasters, they all contain only seed facts and no learned facts. The only prior work we are aware of is by [78], who use manually created patterns to mine children cardinalities from Wikipedia. They showed that with 30 manually crafted patterns and simple filters it is possible to extract 86,227 children-cardinality-assertions with a precision of 94.3%. Our work generalizes upon this, developing an automated technique for extracting relation cardinalities.

8.8. Summary

In Chapter 7, we have discussed how completeness statements can be created manually via CORNER and COOL-WD. To improve the scalability, one may rely on an automated method of extracting relation cardinalities from text, which can then be matched with the number of relation values in a KB to generate completeness statements. In this chapter, we have introduced the problem of extracting relation cardinalities from text, discussed the challenges that set it apart from standard information extraction, and developed a CRF-based distantly-supervised technique for the extraction. There are several avenues to extend this work. On the technical side, the present work does not consider instances with no facts in training (due to their overwhelming proportion), and is thus not suited to predict zero cardinality (like Angela Merkel having no children).

Furthermore, compositionality is only explored within sentences, while in reality it appears also spread over multiple sentences. Taking this even further, one might even look at multiple sources, which may have different timestamps, and use techniques from truth discovery and data fusion to retrieve most likely values in the case of conflicts.

A third direction is to go towards constraints and statistical reasoning. Ordinal number like in *“His second wife”* are ignored by our method, but are valuable clues as they set lower bounds on relation cardinalities. Similarly, the number of brothers and sisters should add up to the number of siblings, having 80 band members is uncommon, or sports teams normally have fewer coaches than players. Learning such constraints, or exploiting them in the consolidation part of relation cardinality extraction, could be fruitful to further improve precision and recall of the present method.

Chapter 9

Discussion

Here we discuss issues related to our completeness management framework: acquisition of completeness information and compatibility with advanced RDF features.

9.1. Acquisition of Completeness Information

Sources of Completeness Information. Our framework relies on the availability of machine-readable completeness information. We found a widespread interest in collecting completeness information in various forms. For instance, Wikipedia provides a template for adding completeness statements[1] and contains over 14,000 pages with the keywords 'complete list of' and 'list is complete;' IMDb has at least 24,000 verified statements about the completeness of cast and crew;[2] and OpenStreetMap has around 2,200 pages featuring completeness status.[3] The techniques we develop may serve as an incentive to standardize such information and to make it available in RDF, since then not only is such information useful for managing data quality, but also for assessing query quality in terms of completeness and soundness.

Ideas for approaches to automating the generation of completeness information are collected in [98], in addition to our idea that is based on relation cardinality extraction as in Chapter 8. Our COOL-WD demonstrator (see Chapter 7) for managing and consuming standardized completeness information of Wikidata is available

[1]`https://en.wikipedia.org/wiki/Template:Complete_list`

[2]`http://www.imdb.com/interfaces`

[3]For instance, see `http://wiki.openstreetmap.org/wiki/Abingdon`

at `http://cool-wd.inf.unibz.it`, which currently stores over 10,000 real completeness statements.

Correctness of Completeness Statements. Any inference is only as correct as the used antecedents. If owners of data sources can add completeness annotations by themselves, incorrect completeness annotations can 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 based on the completeness assertions X, Y and Z over the data source W, given by agents A and B on date D"). Such provenance information can then serve as a basis for trust determination over conclusions. We refer, e.g., to [9, 48, 63] for work about trust and provenance.

Another view on correctness is that analogously to completeness statements, one can also formulate correctness statements, and use them for annotating query answers with correctness information. This was already observed by Motro [80]. While both completeness and correctness are important issues on the Semantic Web, we focus here on completeness. We believe that people in general publish data that they think is correct, while they are aware that not all data is complete.

9.2. Compatibility with Advanced RDF Features

Blank Nodes. The use of blank nodes in RDF has been a controversial topic in the Semantic Web community [2, 1]. In Linked Data applications, blank nodes add complexity to data processing and data interlinking due to the local scope of their labels [50, 37]. With respect to SPARQL, there are semantic mismatches with the RDF semantics of blank nodes, e.g., when COUNT and NOT-EXISTS features are employed [56]. Nevertheless, blank nodes are used in practice to some degree: (*i*) for modeling unknown nulls [56, 31], and (*ii*) for modeling n-ary relations as auxiliary instances in reification [84].

For the former usage, it will be a contradiction if something is complete but unknown, as we argue that completeness statements should capture only "known and complete" information. Say, one may state that a graph is complete for triples of the form $(john, child, ?y)$, while the graph contains the triple $(john, child, \_\!:\!\mathtt{b})$, indicating that

John is complete for his unknown child, which does not really make sense. Nevertheless, a graph with completeness statements may still have blank nodes as long as they are not captured by the statements.

For the latter case, skolemization as a way to systematically replace blank nodes with fresh, skolem IRIs may be leveraged with almost interchangeable behavior [25, 49, 55], except that skolem IRIs have a global scope instead of a local scope. This way, completeness statements can capture n-ary relation information encoded originally with blank nodes, and completeness reasoning (which involves SPARQL queries) behaves well (i.e., no semantic mismatches [56]). Nevertheless, in practice Semantic Web developers tend to directly use IRIs instead of blank nodes for representing auxiliary resources, as demonstrated by Wikidata [37].[4]

RDFS Extension. RDFS [16] adds light-weight semantics to describe the structure and interlinking of data, usually sufficient for Linked Data publishers [50]. Main RDFS inference capabilities consist of class and property hierarchies, as well as property domains and ranges [50, 81], which are widely used in practice [89]. Darari et al. [27] formalized the incorporation of RDFS in data-agnostic completeness reasoning. Moreover, our CORNER system (see Section 7.1) demonstrates such RDFS incorporation in the data-agnostic settings.

Using a similar technique as in [27], it is also relatively easy to extend our data-aware completeness reasoning framework with the RDFS semantics. The idea is that we strengthen our syntactic characterization of computing the *epg* operator (see Subsection 3.2.1) via the closure operation wrt. RDFS ontologies [81]. More precisely, in the crucial part, the closure has to be computed before and after the $T_{\mathbf{C}}$ operation over $\tilde{P} \cup G$. Also, the evaluation of the crucial part needs to be done over the materialized graph G wrt. the RDFS ontology. As for query soundness checking, a similar procedure based on RDFS closure needs to be employed as well. For pattern soundness reasoning, we include the closure computation in the query set containment checking for Non-Redundant Form (NRF), and in the query completeness checking (as in Proposition 5.4). For answer soundness checking, we can simply rely on the data-aware completeness checking with RDFS incorporation we just sketched. In summary, the addi-

[4]For instance, the resource IRI of Wikidata for the marriage between Donald Trump and Ivana Trump is `http://www.wikidata.org/entity/statement/q22686-f813c208-48b2-9a72-3c53-cdaed80518d2`.

tion of the closure computation ensures that the semantics of RDFS is incorporated in the reasoning, while not increasing the complexity as the RDFS closure computation can be done in PTIME [81].

Chapter 10

Conclusions and Future Directions

The objective of this thesis is to study the problem of completeness for RDF data sources. We conclude by summing up the results, relating them with the research hypotheses (as given in Section 1.1), and discussing future work.

10.1. Summary of the Results

Our study was motivated by the question: How complete is Semantic Web data? In Chapter 2, we formalized the notion of completeness over parts of RDF data, and introduced completeness statements as a means to capture partial completeness. Having completeness statements opens up the possibility of checking query completeness. We distinguished between two problems of query completeness entailment, depending on whether the data graph is taken into account: data-agnostic completeness entailment and data-aware completeness entailment.

In Chapter 3, we motivated and formalized data-aware completeness entailment. Furthermore, we devised a technique to check if such entailment holds and studied the complexity of the entailment problem. We identified two different fragments of completeness statements: SP-statements and no-value statements. While SP-statements are more suited to capturing completeness for entity-centric, crowd-sourced RDF data sources, no-value statements can be leveraged to tackle the problem of non-existent information and query emptiness. With data-aware reasoning, we have shown that previously incomplete queries by data-agnostic reasoning can become complete. We verified our research hypothesis that the incorporation of available graphs gives a stronger, fine-grained query completeness assessment.

In Chapter 4, we developed optimization techniques for completeness reasoning and conducted experimental evaluations to show its feasibility. As for data-agnostic completeness reasoning, we identified the constant-relevance principle to reduce the number of completeness statements in the reasoning, and investigated various index structures for the retrieval of constant-relevant statements. As for data-aware completeness reasoning, we relied on completeness templates to organize completeness statements, enabling simultaneous processing of the statements, and partial matching that is used to rule out irrelevant completeness templates. Our experimental evaluations over both reasoning problems showed that our optimizations provided a speed-up over unoptimized ones, where the average runtime was below a millisecond for data-agnostic reasoning, and 140 ms for data-aware reasoning in realistic cases. Wrt. the research hypothesis, not only is query completeness checking time comparable to query evaluation time, but it is even faster for data-agnostic settings, thanks to our optimizations. For data-aware settings, however, the completeness checking was slower than query evaluation, though in absolute scale still relatively fast.

In Chapter 5, we studied the problem of soundness for SPARQL queries with negation. We distinguished between two variants of the problem, that is, pattern soundness and answer soundness. We approached the problem via reduction to completeness entailment, and thus confirmed our research hypothesis. We also provided experimental evidence of the feasibility of our soundness reasoning.

In Chapter 6, we extended completeness statements with time. More specifically, we formalized the notion of completeness of parts of data up to some point of time, and query completeness with time. We introduced the guaranteed completeness date (GCD): the latest date on which complete query results are ensured to be included in the actual query results. We developed an algorithm to find such GCD, which is optimal in the sense that timestamped completeness statements are considered at most once. In this regard, temporal completeness analysis can be performed with just little additional cost (in comparison to non-temporal completeness analysis), which confirmed our hypothesis.

To show how our theoretical completeness framework can be used in practice, in Chapter 7 we developed two demonstration systems: CORNER and COOL-WD. They showcased how the completeness life cycle, consisting of the creation, view, update, and consumption of completeness statements, can be facilitated. CORNER demonstrated a completeness statement hub over multiple RDF data sources, that supports data-agnostic query completeness checking with RDFS ontologies and federated rewriting. COOL-WD demonstrated functionalities for managing and consuming completeness information over Wikidata. With COOL-WD one can create SP-statements about Wikidata entities. On the consumption side, COOL-WD provided features such as data completion tracking, completeness analytics, and query completeness assessment with diagnostics. As also discussed in the chapter, the development of the above systems made use of existing Semantic Web libraries (e.g., Apache Jena). Hence, there was only little development overhead, and our research hypothesis was verified.

As an alternative to collecting completeness statements, we investigated an automated method for relation cardinality extraction in Chapter 8. Such cardinality information can be leveraged to generate completeness statements in the following way: when the value count of an entity's relation in a KB matches the cardinality information of the entity's relation found in text, then a completeness statement of the entity's relation for that KB can be generated. We focused on extracting relation cardinalities on Wikipedia, and developed an extraction method based on conditional random fields (CRF) with distant-supervision. We analyzed three aspects that make relation cardinality extraction challenging: quality of training data, compositionality, and linguistic variance, and showed that our method can achieve precision scores of up to 84%. Given this effectiveness, our research hypothesis was largely confirmed: cardinality information in natural language texts can be extracted to provide hints about completeness information for RDF data sources.

In Chapter 9 we discussed two crucial aspects of our completeness management framework. For the aspect of acquisition of completeness information, we outlined possible sources of completeness statements and raised the issue of correctness of completeness statements. For the aspect of compatibility with advanced RDF features, we discussed how our completeness framework deals with blank nodes and RDFS.

10.2. Future Work

The results of this thesis can be extended in several ways. First, it is of interest to see how our completeness management framework can be extended with OWL [52]. OWL provides inferences that go beyond RDFS such as class disjointness, existential and universal quantification of property restrictions, and property chains. Furthermore, there is also the enumeration of individuals which is similar to completeness statements. We are interested to know to which extent OWL features can enrich our completeness framework, and also, which OWL profiles (i.e., OWL 2 EL, OWL 2 QL, OWL 2 RL) are the most suitable to extend our work.

Next, while we have addressed the BGP fragment of SPARQL for the completeness problem and the BGP fragment with several `NOT EXISTS` negations for the soundness problem, we are curious about enriching our query fragments with more constructors. The `OPTIONAL` constructor, for instance, allows parts of graph patterns to be optionally matched to graphs. For data-agnostic completeness reasoning, Darari et al. [27] have investigated the inclusion of the `OPTIONAL` constructor. Nevertheless, it is still open how the `OPTIONAL` constructor may behave in data-aware completeness reasoning and soundness reasoning. Also, while the current completeness statements are constructed using BGPs, one might wonder what happens if richer constructors are added, to enable statements like "Complete for all UniBZ students who were born after 1991 and who do not speak German."

From the practical side, one future direction is to study how (Semantic) Web data publishers and users perceive the problem of completeness, and how they want to benefit from data completeness. Extensive case studies may be conducted in various application domains like healthcare, economics, or education. The purpose is to analyze whether our completeness framework is sufficient or not for their requirements, and if not, on which side it can be improved. In regard to completeness statement availability, usability evaluations over our completeness demonstration systems can be conducted, with the aim to increase potential user engagement. Moreover, our automated technique for relation cardinality extraction may be reinforced to handle more relations with a better precision, by considering, for instance, named entity recognition, coreference resolution, and knowledge base integration.

Bibliography

[1] Richard Cyganiak: Blank nodes considered harmful. `http://richard.cyganiak.de/blog/2011/03/blank-nodes-considered-harmful/`. Accessed: 2017-01-15.

[2] semantic-web@w3.org Mail Archives: a blank node issue. `https://lists.w3.org/Archives/Public/semantic-web/2011Mar/0017.html`. Accessed: 2017-01-15.

[3] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[4] Maribel Acosta, Elena Simperl, Fabian Flöck, and Maria-Esther Vidal. HARE: A hybrid SPARQL engine to enhance query answers via crowdsourcing. In *K-CAP*, 2015.

[5] Shqiponja Ahmetaj, Magdalena Ortiz, and Mantas Simkus. Polynomial datalog rewritings for ontology mediated queries with closed predicates. In *Proceedings of the 10th Alberto Mendelzon International Workshop on Foundations of Data Management, Panama City, Panama, May 8-10, 2016*, 2016.

[6] Keith Alexander, Richard Cyganiak, Michael Hausenblas, and Jun Zhao. *Describing Linked Datasets with the VoID Vocabulary.* W3C Interest Group Note, 3 March 2011. Retrieved Feb 1, 2015 from `http://www.w3.org/TR/2011/NOTE-void-20110303/`.

[7] Anastasia Analyti, Grigoris Antoniou, Carlos Viegas Damásio, and Gerd Wagner. Extended RDF as a semantic foundation of rule markup languages. *J. Artif. Intell. Res. (JAIR)*, 32:37–94, 2008.

[8] Mario Arias, Javier D. Fernández, Miguel A. Martínez-Prieto, and Pablo de la Fuente. An Empirical Study of Real-World SPARQL Queries. In *Proceedings of the 1^{st} International Work-*

shop on Usage Analysis and the Web of Data (USEWOD'11), 2011.

[9] Donovan Artz and Yolanda Gil. A survey of trust in computer science and the Semantic Web. *J. Web Sem.*, 5(2):58–71, 2007.

[10] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. *DBpedia: A nucleus for a web of open data*. Springer, 2007.

[11] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. Querying RDF streams with C-SPARQL. *SIGMOD Record*, 39(1):20–26, 2010.

[12] Carlo Batini and Monica Scannapieco. *Data and Information Quality - Dimensions, Principles and Techniques*. Data-Centric Systems and Applications. Springer, 2016.

[13] David Becker, Trish Dunn King, and Bill McMullen. Big data, big data quality problem. In *2015 IEEE International Conference on Big Data, Big Data 2015, Santa Clara, CA, USA, October 29 - November 1, 2015*, pages 2644–2653, 2015.

[14] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia – A Crystallization Point for the Web of Data. *Journal of Web Semantics*, 7(3), 2009.

[15] Christian Bizer and Andreas Schultz. The Berlin SPARQL Benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24, 2009.

[16] Dan Brickley and R. V. Guha. *RDF Schema 1.1*. W3C Recommendation, 25 February 2014. Retrieved Jan 10, 2017 from `http://www.w3.org/TR/2014/REC-rdf-schema-20140225/`.

[17] Jean-Paul Calbimonte, Óscar Corcho, and Alasdair J. G. Gray. Enabling ontology-based access to streaming data sources. In *International Semantic Web Conference*, pages 96–111, 2010.

[18] Arun Chaganty and Percy Liang. How Much is 131 Million Dollars? Putting Numbers in Perspective with Compositional Descriptions. In *ACL*, pages 578–587, August 2016.

[19] Ashok K. Chandra and Philip M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *Proceedings of the 9th ACM Symposium on Theory of Computing (STOC'77)*, 1977.

[20] Xu Chu, John Morcos, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. KATARA: A data cleaning sys-

tem powered by knowledge bases and crowdsourcing. In *ACM SIGMOD*, 2015.

[21] Keith L. Clark. Negation as Failure. In *Logic and Data Bases*, pages 113–141, 1978.

[22] E. F. Codd. Extending the database relational model to capture more meaning. *ACM Trans. Database Syst.*, 4(4):397–434, 1979.

[23] Edgar F. Codd. Relational completeness of data base sublanguages. *In: R. Rustin (ed.): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987, San Jose, California*, 1972.

[24] Mark Craven and Johan Kumlien. Constructing biological knowledge bases by extracting information from text sources. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 77–86, 1999.

[25] Richard Cyganiak, David Wood, and Markus Lanthaler, editors. *RDF 1.1 Concepts and Abstract Syntax.* W3C Recommendation, 25 February 2014. Retrieved Jan 15, 2017 from `https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/`.

[26] Fariz Darari. Representing and querying negative knowledge in RDF. In *The Semantic Web: ESWC 2013 Satellite Events - ESWC 2013 Satellite Events, Montpellier, France, May 26-30, 2013, Revised Selected Papers*, pages 275–276, 2013.

[27] Fariz Darari, Werner Nutt, Giuseppe Pirrò, and Simon Razniewski. Completeness statements about RDF data sources and their use for query answering. In *ISWC*, 2013.

[28] Fariz Darari, Werner Nutt, Giuseppe Pirrò, and Simon Razniewski. Completeness management for RDF data sources. *TWEB*, 12(3):18:1–18:53, 2018.

[29] Fariz Darari, Werner Nutt, Simon Razniewski, and Sebastian Rudolph. Completeness and soundness guarantees for conjunctive SPARQL queries over RDF data sources with completeness statements. *Semantic Web*, Pre-press(Pre-press):1–42, 2019.

[30] Fariz Darari, Radityo Eko Prasojo, and Werner Nutt. CORNER: A completeness reasoner for SPARQL queries over RDF data sources. In *ESWC Demos*, 2014.

[31] Fariz Darari, Radityo Eko Prasojo, and Werner Nutt. Expressing no-value information in RDF. In *Proceedings of the ISWC 2015 Posters & Demonstrations Track co-located with the 14th International Semantic Web Conference (ISWC-2015), Bethle-*

hem, PA, USA, October 11, 2015., 2015.

[32] Fariz Darari, Simon Razniewski, and Werner Nutt. Bridging the semantic gap between RDF and SPARQL using completeness statements. In *Proceedings of the ISWC 2014 Posters & Demonstrations Track a track within the 13th International Semantic Web Conference, ISWC 2014, Riva del Garda, Italy, October 21, 2014.*, pages 269–272, 2014.

[33] Fariz Darari, Simon Razniewski, Radityo Eko Prasojo, and Werner Nutt. Enabling fine-grained RDF data completeness assessment. In *ICWE*, 2016.

[34] Luciano Del Corro and Rainer Gemulla. ClausIE: clause-based open information extraction. In *WWW*, pages 355–366. ACM, 2013.

[35] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In *ACM SIGKDD 2014*, pages 601–610, 2014.

[36] Ivan Ermilov, Jens Lehmann, Michael Martin, and Sören Auer. LODStats: The Data Web Census Dataset. In *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part II*, pages 38–46, 2016.

[37] Fredo Erxleben, Michael Günther, Markus Krötzsch, Julian Mendez, and Denny Vrandecic. Introducing Wikidata to the Linked Data Web. In *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, pages 50–65, 2014.

[38] Christian Fürber and Martin Hepp. Towards a vocabulary for data quality management in semantic web architectures. In *Proceedings of the 2011 EDBT/ICDT Workshop on Linked Web Data Management, Uppsala, Sweden, March 25, 2011*, pages 1–8, 2011.

[39] Luis Galárraga, Simon Razniewski, Antoine Amarilli, and Fabian M. Suchanek. Predicting completeness in knowledge bases. In *Conference on Web Search and Data Mining (WSDM)*, 2017.

[40] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M Suchanek. Fast rule mining in ontological knowledge bases with

AMIE+. *VLDB Journal*, 24(6):707–730, 2015.

[41] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. AMIE: Association Rule Mining under Incomplete Evidence in Ontological Knowledge Bases. In *WWW 2013*, pages 413–422, 2013.

[42] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, USA, 1990.

[43] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.

[44] Claudio Gutierrez, Daniel Hernández, Aidan Hogan, and Axel Polleres. Certain Answers for SPARQL? In *Proceedings of the 10th Alberto Mendelzon International Workshop on Foundations of Data Management, Panama City, Panama, May 8-10, 2016*, 2016.

[45] Claudio Gutiérrez, Carlos A. Hurtado, and Alejandro A. Vaisman. Temporal RDF. In *The Semantic Web: Research and Applications, Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29 - June 1, 2005, Proceedings*, pages 93–107, 2005.

[46] Steve Harris and Andy Seaborne, editors. *SPARQL 1.1 Query Language.* W3C Recommendation, 21 March 2013. Retrieved Feb 1, 2015 from `http://www.w3.org/TR/2013/REC-sparql11-query-20130321/`.

[47] Andreas Harth and Sebastian Speiser. On Completeness Classes for Query Evaluation on Linked Data. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI'12)*, 2012.

[48] Olaf Hartig. Provenance information in the web of data. In *Proceedings of the WWW2009 Workshop on Linked Data on the Web, LDOW 2009, Madrid, Spain, April 20, 2009.*, 2009.

[49] Patrick J. Hayes and Peter F. Patel-Schneider, editors. *RDF 1.1 Semantics.* W3C Recommendation, 25 February 2014. Retrieved Jan 15, 2017 from `https://www.w3.org/TR/2014/REC-rdf11-mt-20140225/`.

[50] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space.* Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool, 2011.

[51] Sven Helmer and Guido Moerkotte. A Performance Study of

Four Index Structures for Set-Valued Attributes of Low Cardinality. *VLDB Journal*, 12(3), 2003.

[52] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph, editors. *OWL 2 Web Ontology Language Primer (Second Edition).* W3C Recommendation, 11 December 2012. Retrieved Jan 1, 2017 from `https://www.w3.org/TR/owl2-primer/`.

[53] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, Edwin Lewis-Kelham, Gerard de Melo, and Gerhard Weikum. 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)*, 2011.

[54] Jörg Hoffmann and Jana Koehler. A New Method to Index and Query Sets. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, 1999.

[55] Aidan Hogan. Skolemising blank nodes while preserving isomorphism. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, pages 430–440, 2015.

[56] Aidan Hogan, Marcelo Arenas, Alejandro Mallea, and Axel Polleres. Everything you always wanted to know about blank nodes. *J. Web Sem.*, 27:42–69, 2014.

[57] Yusra Ibrahim, Mirek Riedewald, and Gerhard Weikum. Making sense of entities and quantities in web tables. In *CIKM*, pages 1703–1712, 2016.

[58] Robin Keskisärkkä and Eva Blomqvist. Event object boundaries in RDF streams. In *Proceedings of the 2nd International Workshop on Ordering and Reasoning, OrdRing 2013, Co-located with the 12th International Semantic Web Conference (ISWC 2013), Sydney, Australia, October 22nd, 2013*, pages 37–42, 2013.

[59] Graham Klyne and Jeremy J. Carroll, editors. *Resource Description Framework (RDF): Concepts and Abstract Syntax.* W3C Recommendation, 10 February 2004. Retrieved Feb 1, 2015 from `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`.

[60] Holger Knublauch and Dimitris Kontokostas, editors. *Shapes Constraint Language (SHACL).* W3C Candidate Recom-

mendation, 11 April 2017. Retrieved May 20, 2017 from https://www.w3.org/TR/2017/CR-shacl-20170411/.

[61] Mitchell Koch, John Gilmer, Stephen Soderland, and Daniel S. Weld. Type-aware distantly supervised relation extraction with linked arguments. In *EMNLP*, pages 1891–1901, 2014.

[62] Taku Kudo. CRF++: Yet another CRF toolkit. *Software available at http://crfpp. sourceforge.net*, 2005.

[63] Timothy Lebo, Satya Sahoo, and Deborah McGuinness, editors. *PROV-O: The PROV Ontology.* W3C Candidate Recommendation, 11 December 2012. Retrieved May 27, 2016 from `https://www.w3.org/TR/2012/CR-prov-o-20121211/`.

[64] Alon Y. Levy. Obtaining complete answers from incomplete databases. In *VLDB*, 1996.

[65] Xiao Ling and Daniel S Weld. Temporal information extraction. In *AAAI*, volume 10, pages 1385–1390, 2010.

[66] Nuno Lopes, Axel Polleres, Umberto Straccia, and Antoine Zimmermann. AnQL: SPARQLing Up Annotated RDFS. In *The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I*, pages 518–533, 2010.

[67] Carsten Lutz, Inanç Seylan, and Frank Wolter. Ontology-based data access with closed predicates is inherently intractable (sometimes). In *IJCAI*, 2013.

[68] Carsten Lutz, Inanç Seylan, and Frank Wolter. Ontology-mediated queries with closed predicates. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 3120–3126, 2015.

[69] Aman Madaan, Ashish Mittal, G Ramakrishnan Mausam, Ganesh Ramakrishnan, and Sunita Sarawagi. Numerical relation extraction with minimal supervision. In *AAAI*, pages 2764–2771, 2016.

[70] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. *ACL (System Demonstrations)*, pages 55–60, 2014.

[71] Frank Manola and Eric Miller, editors. *RDF Primer.* W3C Recommendation, 10 February 2004. Retrieved Jul 31, 2016 from `https://www.w3.org/TR/2004/REC-rdf-primer-20040210/`.

[72] Mausam, Michael Schmitz, Stephen Soderland, Robert Bart, and Oren Etzioni. Open language learning for information extraction. In *EMNLP*, pages 523–534, 2012.

[73] Pablo N. Mendes, Hannes Mühleisen, and Christian Bizer. Sieve: Linked Data quality assessment and fusion. In *Joint EDBT/ICDT Workshops*, 2012.

[74] Bonan Min, Ralph Grishman, Li Wan, Chang Wang, and David Gondek. Distant supervision for relation extraction with an incomplete knowledge base. In *NAACL*, pages 777–782, June 2013.

[75] Mike Mintz, Steven Bills, Rion Snow, and Daniel Jurafsky. Distant supervision for relation extraction without labeled data. In *ACL*, pages 1003–1011, 2009.

[76] Paramita Mirza, Simon Razniewski, Fariz Darari, and Gerhard Weikum. Cardinal virtues: Extracting relation cardinalities from text. In *ACL (Short Papers)*, 2017.

[77] Paramita Mirza, Simon Razniewski, Fariz Darari, and Gerhard Weikum. Enriching knowledge bases with counting quantifiers. In *The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part I*, pages 179–197, 2018.

[78] Paramita Mirza, Simon Razniewski, and Werner Nutt. Expanding WikidataâĂŹs parenthood information by 178%, or how to mine relation cardinalities. *ISWC Posters & Demos*, 2016.

[79] Tom M. Mitchell, William W. Cohen, Estevam R. Hruschka Jr., Partha Pratim Talukdar, Justin Betteridge, Andrew Carlson, Bhavana Dalvi Mishra, Matthew Gardner, Bryan Kisiel, Jayant Krishnamurthy, Ni Lao, Kathryn Mazaitis, Thahir Mohamed, Ndapandula Nakashole, Emmanouil Antonios Platanios, Alan Ritter, Mehdi Samadi, Burr Settles, Richard C. Wang, Derry Tanti Wijaya, Abhinav Gupta, Xinlei Chen, Abulhair Saparov, Malcolm Greaves, and Joel Welling. Never-ending learning. In *AAAI*, pages 2302–2310, 2015.

[80] Amihai Motro. Integrity = Validity + Completeness. *ACM Trans. Database Syst.*, 14(4), 1989.

[81] Sergio Muñoz, Jorge Pérez, and Claudio Gutierrez. Simple and efficient minimal RDFS. *J. Web Sem.*, 7(3):220–234, 2009.

[82] Sebastian Neumaier, Jürgen Umbrich, Josiane Xavier Parreira, and Axel Polleres. Multi-level semantic labelling of numerical

values. In *ISWC*, pages 428–445, 2016.

[83] Nhung Ngo, Magdalena Ortiz, and Mantas Simkus. Closed predicates in description logics: Results on combined complexity. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016.*, pages 237–246, 2016.

[84] Natasha Noy and Alan Rector, editors. *Defining N-ary Relations on the Semantic Web.* W3C Working Group Note, 12 April 2006. Retrieved Jan 10, 2017 from `https://www.w3.org/TR/2006/NOTE-swbp-n-aryRelations-20060412/`.

[85] Thomas Palomares, Youssef Ahres, Juhana Kangaspunta, and Christopher Ré. Wikipedia knowledge graph with DeepDive. In *ICWSM*, pages 65–71, 2016.

[86] Heiko Paulheim. Identifying wrong links between datasets by multi-dimensional outlier detection. In *WoDOOM*, pages 27–38, 2014.

[87] Danh Le Phuoc, Minh Dao-Tran, Josiane Xavier Parreira, and Manfred Hauswirth. A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data. In *International Semantic Web Conference*, pages 370–388, 2011.

[88] Axel Polleres, Cristina Feier, and Andreas Harth. Rules with contextually scoped negation. In *ESWC*, 2006.

[89] Axel Polleres, Aidan Hogan, Renaud Delbru, and Jürgen Umbrich. RDFS and OWL reasoning for linked data. In *Reasoning Web. Semantic Technologies for Intelligent Data Access - 9th International Summer School 2013, Mannheim, Germany, July 30 - August 2, 2013. Proceedings*, pages 91–149, 2013.

[90] Radityo Eko Prasojo, Fariz Darari, Simon Razniewski, and Werner Nutt. 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, COLD@ISWC 2015, Kobe, Japan, October 18, 2016.*, 2016.

[91] Eric Prud'hommeaux and Carlos Buil-Aranda, editors. *SPARQL 1.1 Federated Query.* W3C Recommendation, 21 March 2013. Retrieved Jan 10, 2017 from `https://www.w3.org/TR/sparql11-federated-query/`.

[92] Eric Prudhommeaux and Gavin Carothers, editors. *RDF 1.1*

Turtle. W3C Recommendation, 25 February 2014. Retrieved Jan 1, 2017 from `https://www.w3.org/TR/turtle/`.

[93] Eric Prud'hommeaux and Andy Seaborne, editors. *SPARQL Query Language for RDF*. W3C Recommendation, 15 January 2008.

[94] Andrea Pugliese, Octavian Udrea, and V. S. Subrahmanian. Scaling RDF with time. In *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*, pages 605–614, 2008.

[95] Simon Razniewski, Flip Korn, Werner Nutt, and Divesh Srivastava. Identifying the extent of completeness of query answers over partially complete databases. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 561–576, 2015.

[96] Simon Razniewski and Werner Nutt. Completeness of queries over incomplete databases. *PVLDB*, 4(11):749–760, 2011.

[97] Simon Razniewski and Werner Nutt. Assessing Query Completeness over Incomplete Databases. In *Unpublished manuscript*, 2015.

[98] Simon Razniewski, Fabian M. Suchanek, and Werner Nutt. But what do we actually know? In *AKBC Workshop at NAACL*, 2016.

[99] Raymond Reiter. On closed world data bases. In Hervé Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 55–76. Springer US, Boston, MA, 1978.

[100] Muhammad Saleem, Muhammad Intizar Ali, Aidan Hogan, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo. LSQ: The Linked SPARQL Queries Dataset. In *ISWC*, 2015.

[101] Ognjen Savkovic, Paramita Mirza, Sergey Paramonov, and Werner Nutt. MAGIK: managing completeness of data. In *CIKM Demos*, 2012.

[102] Iztok Savnik. Index Data Structure for Fast Subset and Superset Queries. In *International Cross Domain Conference and Workshop (CD-ARES'13)*, 2013.

[103] Michael Schmidt, Olaf Görlitz, Peter Haase, Günter Ladwig, Andreas Schwarte, and Thanh Tran. FedBench: A Benchmark Suite for Federated Semantic Data Query Processing. In *ISWC*, 2011.

[104] Michael Schmidt, Thomas Hornung, Michael Meier, Christoph Pinkel, and Georg Lausen. SP^{2}Bench: A SPARQL Performance Benchmark. In *Semantic Web Information Management - A Model-Based Perspective*, 2009.

[105] Inanç Seylan, Enrico Franconi, and Jos de Bruijn. Effective Query Rewriting with Ontologies over DBoxes. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 923–925, 2009.

[106] Robert Speer and Catherine Havasi. Representing General Relational Knowledge in ConceptNet 5. In *LREC*, 2012.

[107] Jannik Strötgen and Michael Gertz. Heideltime: High quality rule-based extraction and normalization of temporal expressions. In *SemEval Workshop*, pages 321–324, 2010.

[108] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. YAGO: a core of semantic knowledge. *WWW*, pages 697–706, 2007.

[109] Mihai Surdeanu, Julie Tibshirani, Ramesh Nallapati, and Christopher D. Manning. Multi-instance multi-label learning for relation extraction. In *ACL*, pages 455–465, 2012.

[110] Jonas Tappolet and Abraham Bernstein. Applied temporal RDF: efficient temporal querying of RDF data with SPARQL. In *The Semantic Web: Research and Applications, 6th European Semantic Web Conference, ESWC 2009, Heraklion, Crete, Greece, May 31-June 4, 2009, Proceedings*, pages 308–322, 2009.

[111] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

[112] Richard Y. Wang and Diane M. Strong. Beyond accuracy: What data quality means to data consumers. *J. of Management Information Systems*, 12(4):5–33, 1996.

[113] Wei Xu, Raphael Hoffmann, Le Zhao, and Ralph Grishman. Filling knowledge base gaps for distant supervision of relation extraction. In *ACL (short paper)*, pages 665–670, August 2013.

[114] Amrapali Zaveri, Anisa Rula, Andrea Maurino, Ricardo Pietrobon, Jens Lehmann, and Sören Auer. Quality assessment for Linked Data: A Survey. *Semantic Web*, 7(1):63–93, 2016.

[115] Jiawei Zhang, Jianhui Chen, Junxing Zhu, Yi Chang, and Philip S Yu. Link prediction with cardinality constraint. In

WSDM, 2017.

[116] Justin Zobel, Alistair Moffat, and Ron Sacks-Davis. An Efficient Indexing Technique for Full-Text Databases. In *Proceedings of the 18th International Conference on Very Large Data Bases (VLDB'92)*, 1992.

Appendix A

Prefix Declarations

Here we provide in Turtle syntax [92] the prefix declarations of the RDF snippets in this thesis. The prefixes can be adapted accordingly for the SPARQL snippets in this thesis.

```
@prefix c: <http://completeness.inf.unibz.it/ns#> .
@prefix coolwd: <http://cool-wd.inf.unibz.it/resource/> .
@prefix dbo: <http://dbpedia.org/ontology/> .
@prefix dbp: <http://dbpedia.org/resource/> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix ex: <http://example.org/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix lv: <http://linkedmdb.org/void/> .
@prefix no: <http://completeness.inf.unibz.it/no-value#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix prov: <http://www.w3.org/ns/prov#> .
@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 sp: <http://spinrdf.org/sp#> .
@prefix spv: <http://completeness.inf.unibz.it/sp-vocab#> .
@prefix void: <http://rdfs.org/ns/void#> .
@prefix wd: <http://www.wikidata.org/entity/> .
@prefix wdt: <http://www.wikidata.org/prop/direct/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
```

$$Compl((farizPhDThesis, hasPage, ?page)).$$