Semantic Web 0 (0) 1
IOS Press

Completeness and Soundness Guarantees for Conjunctive SPARQL Queries over RDF Data Sources with Completeness Statements[1]

Fariz Darari [a,*], Werner Nutt [b], Simon Razniewski [c], Sebastian Rudolph [d]
[a] *Faculty of Computer Science, Universitas Indonesia, Indonesia*
E-mail: fariz@cs.ui.ac.id
[b] *Faculty of Computer Science, Free University of Bozen-Bolzano, Italy*
E-mail: Werner.Nutt@unibz.it
[c] *Max-Planck-Institute for Informatics, Germany*
E-mail: srazniew@mpi-inf.mpg.de
[d] *Faculty of Computer Science, TU Dresden, Germany*
E-mail: sebastian.rudolph@tu-dresden.de

Abstract. RDF generally follows the open-world assumption: information is incomplete by default. Consequently, SPARQL queries cannot retrieve with certainty complete answers, and even worse, when they involve negation, it is unclear whether they produce sound answers. Nevertheless, there is hope to lift this limitation. On many specific topics (e.g., children of Trump, Apollo 11 crew, EU founders), RDF data sources contain complete information, a fact that can be made explicit through completeness statements. In this work, we leverage completeness statements over RDF data sources to provide guarantees of completeness and soundness for conjunctive SPARQL queries. We develop a technique to check whether query completeness can be guaranteed by taking into account also the specifics of the queried graph, and analyze the complexity of such checking. For queries with negation, we approach the problem of query soundness checking, and distinguish between answer soundness (i.e., is an answer of a query sound?) and pattern soundness (i.e., is a query as a whole sound?). We provide a formalization and characterize the soundness problem via a reduction to the completeness problem. We further develop heuristic techniques for completeness checking, and conduct experimental evaluations based on Wikidata, a prominent, real-world knowledge base, to demonstrate the feasibility of our approach.

Keywords: Data quality, data completeness, query completeness, query soundness, RDF, SPARQL

1. Introduction

Over the Web, we are witnessing a growing amount of data available in RDF. As of July 2018, the LOD Cloud[2] has recorded over 1,200 RDF data sources, covering a wide range of application domains from government to life sciences. RDF follows the open-world assumption (OWA), assuming that data is inherently incomplete [2]. Yet, given such a large quantity of RDF data, one might wonder if it is complete for some topics. As an illustration, consider Wikidata, a collaborative knowledge base (KB) whose content is made available in RDF [3]. For data about the movie Reservoir Dogs, Wikidata is incomplete,[3] as it is missing the fact that Michael Sottile was acting in that movie.[4] On the other hand, for data about the European Union (EU), Wikidata actually stores *all* of its

[1] This paper is an extended and revised version of Darari et al. [1].
[*] Corresponding author. E-mail: fariz@cs.ui.ac.id.
[2] http://lod-cloud.net/
[3] https://www.wikidata.org/wiki/Q72962 (as of July 31, 2018)
[4] See, e.g., http://www.imdb.com/title/tt0105236/fullcredits

founding members,[5] as shown in Figure 1. Nevertheless, the figure does not provide any indicator about completeness, leaving the user undecided whether the presented facts about the EU founders are complete or not.

Fig. 1. Wikidata is complete for all EU founding members

The incorporation of completeness information can help users assess the quality of data. Over the Web, completeness information is in fact already available in various forms. For instance, Wikipedia provides a template for adding completeness annotations for lists[6] and contains about 15,000 pages with the keywords 'complete list of' and 'list is complete'; IMDb offers around 52,000 editor-verified (natural language) statements about the completeness of cast and crew;[7] and OpenStreetMap has around 2,200 pages featuring completeness status information.[8] For RDF data, such information about completeness is particularly crucial due to RDF's incomplete nature. In [4], Darari et al. proposed completeness statements, metadata to specify which parts of an RDF data source are complete: for a complete part all facts that hold in reality are captured by the data source. They also provided an RDF representation of completeness statements, making them machine readable. The availability of explicit completeness information opens up the possibility of specialized applications for data source curation, discovery, analytics, and so forth. Moreover, it can also benefit data access over RDF data sources, mainly done via SPARQL queries [5]. More specifically, the quality of query answers can also be made more transparent given that now we know the quality of data sources with regard to completeness.

[5] https://europa.eu/european-union/about-eu/history/
[6] https://en.wikipedia.org/wiki/Template:Complete_list
[7] E.g., see http://www.imdb.com/title/tt0105236/fullcredits
[8] For instance, see http://wiki.openstreetmap.org/wiki/Abingdon

Query Completeness When data sources are enriched with completeness information, the question naturally arises as to whether queries can be answered also completely. Intuitively, queries that are evaluated only over parts of data captured by completeness statements, are guaranteed to be complete. Consider the query "Give the EU founders" over Wikidata:[9]

```
PREFIX  wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>

SELECT * WHERE {
 wd:Q458 wdt:P112 ?c } # EU founder ?c
```

Without completeness information, evaluating the query would give only query answers, for which we do not know the completeness. By having the statement that Wikidata is complete for the EU founders, we can then guarantee that the answers are complete. Darari et al. [4, 6] characterized such reasoning, that is, checking whether queries can be guaranteed to be complete by completeness statements. Nonetheless, this approach is limited in the sense that the specifics of the queried graph are not taken into account in the completeness reasoning.

Let us give an example, illustrating this limitation. Suppose that in addition to the statement about all EU founders, we also have the statements that Wikidata is complete for the official languages of the following countries: Belgium, France, Germany, Italy, Luxembourg, and the Netherlands. Let us now consider the query "Give the EU founders and their official languages." We show that the query can be answered completely by applying a *data-aware* approach: we enumerate the complete EU founders stored in Wikidata, and for each of them, we are complete for its languages. On the other hand, the *data-agnostic* approach from Darari et al. [4, 6] would fail to capture the query completeness: it can be that all the EU founders are completely different than those in Wikidata, and thus, having completeness statements about the six countries above could not help in the reasoning. We argue that data-aware reasoning can provide more fine-grained insights over the completeness of query answers, which otherwise cannot be captured by relying only on the data-agnostic approach.

[9] Wikidata has internal identifiers for resources, as shown in the SPARQL query example.

Query Soundness While it is rather obvious to see that completeness information may guarantee query completeness, one might wonder if such completeness information can also be leveraged to check the soundness of query answers. Indeed, for the positive fragment of SPARQL, the soundness of query answers trivially holds, thanks to monotonicity. Now let us consider queries with negation. There are different ways to express negation in SPARQL, either by the explicit syntactic constructs MINUS and FILTER NOT EXISTS, which are available in SPARQL 1.1, or by combining OPTIONAL patterns with a check for unbound variables, as was the method of choice in SPARQL 1.0. The meaning of such queries on the Semantic Web has always been dubious (see, e.g., W3C mailing list discussions in [7] and [8]). The evaluation of SPARQL queries that include negation relies on the absence of some information. On the other hand, RDF, which follows the OWA, regards missing information as undecided, that is, it is unknown whether the missing information is false. Given this situation, answers to queries with negation can *never* be assured to be sound.

Completeness information can tackle the problem of query soundness. To illustrate this, consider asking for "countries that are not EU founders" over Wikidata:

```
PREFIX  wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>

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

The answers include Spain (= `wd:Q29`). Without any completeness information about Wikidata, we cannot be sure about its soundness: assume Spain were an EU founder, but this information were missing from the data. In that case, Spain is not a correct answer. In reality, the EU founders are *exactly* as shown before in Figure 1. Knowing this guarantees that Spain is *not* an EU founding country. What we can observe here is that negation in SPARQL, due to its inherent *non-monotonicity*, may lead to the problem of judging answer soundness: adding new information may invalidate an answer. *Soundness* of answers is ensured, however, if we know that the parts of the data, over which the negated parts of a query range, are complete (i.e., not open-world).

Contributions An earlier version of our ideas on query completeness checking was published in the Proceedings of the International Conference on Web Engineering [1]. In that work, we provided a formalization of the data-aware completeness entailment problem and developed a sound and complete algorithm for the entailment problem. The SPARQL fragment considered in that work is the conjunctive fragment, which is the core fragment underlying all extensions [9, 10]. The present paper significantly extends the previous work in the following ways:

1. we formulate the soundness problem for conjunctive SPARQL queries with negation in the presence of completeness information, and distinguish between answer soundness (i.e., is an answer of a query sound?) and pattern soundness (i.e., is a query as a whole sound?);
2. we provide a full characterization of both the answer and pattern soundness problem via a reduction to the completeness problem;
3. we identify the bottlenecks of the completeness reasoning techniques from [1] and develop heuristic techniques for completeness reasoning;
4. we provide experimental evaluations based on Wikidata, a prominent, real-world data source to study the effectiveness of the proposed heuristics (and their interplay) and to validate the feasibility of our approach; and
5. we provide a comprehensive complexity analysis of the completeness and soundness entailment problem, include the proofs of all theorems as well as more recent related work, and improve the presentation of the theoretical parts.

A poster by Darari et al. [11] contained a result that can be interpreted as a sufficient condition for soundness in the data-agnostic setting. We now provide a distinction between so-called pattern soundness and answer soundness, a full characterization by means of a reduction to completeness checking, as well as a complexity analysis and experimental evaluations for the soundness problem.

Organization The rest of the article is organized as follows. Section 2 provides some background about RDF and SPARQL, as well as completeness statements. Section 3 motivates and formalizes the problem of query completeness and query soundness. In Section 4, we first introduce formal notions capturing aspects of completeness, then present an algorithm for completeness entailment checking based on those notions, and conclude with a complexity analysis of the completeness entailment problem. We give a characterization of the two problem variants of query soundness, that is, answer soundness and pattern soundness,

in Section 5. We describe our heuristic techniques for completeness checking in Section 6, and report on our experimental evaluations for query completeness and query soundness checking in Section 7. Related work is presented in Section 8. Section 9 provides a discussion of our framework, while Section 10 gives conclusions and future work. Proofs are provided in the appendices.

2. Preliminaries

In this section, we introduce basic notions of RDF and SPARQL, and provide a formalization of completeness statements. We adopt the formalization of RDF and SPARQL as in [12] and base ourselves on the formalization of completeness as in [4, 6].

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. An *RDF graph* (or simply, graph) G is a finite set of triples $(s, p, o) \in I \times I \times (I \cup L)$. For simplicity, we omit namespaces in the abstract representation of RDF graphs.

The standard RDF query language is SPARQL [5]. At the core of SPARQL lie triple patterns, which resemble triples, except that in each position also variables are allowed. A *basic graph pattern* (BGP) is a set of triple patterns. A mapping μ is a partial function $\mu\colon V \to I \cup L$. The operator $dom(\mu)$ returns the set of all variables that are mapped in μ. We define the mapping with the empty domain as the *empty mapping* $\mu_\emptyset$. The operator $var(P)$ denotes the set of variables that are in a BGP P. Given a BGP P, μP denotes the BGP obtained by replacing variables in P with terms according to μ. Evaluating P over a graph G gives the set of mappings $[\![P]\!]_G = \{\mu \mid \mu P \subseteq G \text{ and } dom(\mu) = var(P)\}$.

In the following, we define variable freezing, an important notion for the characterizations of completeness and soundness entailment in the subsequent sections.

Definition 1 (Variable Freezing)**.** *For a BGP P, the* freeze mapping $\tilde{id}$ *maps each variable $?v$ in P to a fresh IRI $\tilde{v}$. With this mapping, we construct the* prototypical graph $\tilde{P} := \tilde{id}\,P$ *to represent any possible graph that can satisfy the BGP P. The* melt mapping $\tilde{id}^{-1}$ *undoes the freezing by substituting the fresh IRIs with the original variables.*[10]

Example 1. Consider the query "Give the founding members of the EU" as introduced in Section 1. The query's BGP can be written as: $\{(EU, founder, ?c)\}$. The freeze mapping of the BGP is $\tilde{id} = \{?c \mapsto \tilde{c}\}$, and the prototypical graph is $\{(EU, founder, \tilde{c})\}$.

The standard query type of SPARQL is the `SELECT` query, which has the abstract form $Q = (W, P)$, where P is a BGP and $W \subseteq var(P)$. The evaluation $[\![Q]\!]_G$ over a graph G is obtained by projecting the mappings in $[\![P]\!]_G$ to W. We study only `SELECT` queries where $W = var(P)$. In this regard, there is no distinction between bag and set semantics (that is, both semantics coincide since duplicates cannot occur). Beside SPARQL `SELECT`, in our formalization later on we will rely on `CONSTRUCT` queries. Given two BGPs P_1 and P_2 where $var(P_1) \subseteq var(P_2)$, a `CONSTRUCT` query has the abstract form $(\texttt{CONSTRUCT}\ P_1\ P_2)$. Evaluating a `CONSTRUCT` query over G yields a graph where P_1 is instantiated with all the mappings in $[\![P_2]\!]_G$.

SPARQL with Negation SPARQL queries can also include negation. We introduce notation that is concise and more convenient for our purposes than the original SPARQL syntax [5]. A `NOT-EXISTS` pattern is constructed by negating a BGP using '$\neg\exists$'. A *graph pattern* P, as used throughout this paper, 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. The evaluation $[\![P]\!]_G$ of a graph pattern P over a graph G produces a set of mappings and is defined in [5] as: $\{\mu \in [\![P^+]\!]_G \mid \forall P_i \in P^- . [\![\mu P_i]\!]_G = \emptyset\}$. As for the fragment with negation, we define the corresponding `SELECT` query as $Q = (var(P^+), P)$, where P is a graph pattern.

Example 2. Consider the query with negation, "Give countries that are not EU founders." The graph pattern of the query can be written as follows: $\{(?c, a, country)$,

[10] Technically, this definition is ambiguous, since there are infinitely many ways to choose the fresh IRIs and therefore there are infinitely many such mappings and corresponding prototypical graphs. In addition, the choice depends on the pattern P so that a correct notation would include a subscript P. To keep our formalism slim, we do not specify how to choose the IRIs, nor do we introduce the subscript, which would be inessential details for the arguments in the paper.

$\neg\exists\{(EU, founder, ?c)\}\}$. The positive part of the query is $\{(?c, a, country)\}$, whereas the negative part is $\{\{(EU, founder, ?c)\}\}$.

We refer to the SPARQL fragment where positive and negative parts of queries are constructed from BGPs as the conjunctive SPARQL fragment with negation. Note that BGPs are the basic building blocks underlying other SPARQL extensions [9, 10] as well.

2.2. Completeness Statements

We want to formalize a mechanism for specifying which parts of a data source are complete. When talking about the completeness of a data source, one implicitly compares the information available in the source with a possible state of the source that contains all information that holds in the real world. We model this situation with a pair of graphs: one graph is the available, possibly incomplete state, while another stands for an ideal, conceptual complete reference, which contains the available graph. In this work, we only consider data sources that may miss information, but do not contain wrong information.

Definition 2 (Extension Pair)**.** *An* extension pair *is a pair (G, G') of two graphs, where $G \subseteq G'$. We call G the* available *graph and G' the* ideal *graph.*

In an application, the state stored in an RDF data source is our actual, available graph, which consists of a part of the facts that hold in reality. The full set of facts that constitute the ideal state are, however, unknown. Nevertheless, an RDF data source can be complete for some parts of the reality. In order to make assertions in this regard, we now introduce completeness statements, as meta-information about the extent to which the available state captures the ideal state. We adopt the unconditional version of the completeness statements defined in [4].

Definition 3 (Completeness Statement)**.** *A* completeness statement *C has the form $Compl(P_C)$, where P_C is a non-empty BGP.*

For example, we express that a data source is complete for all triples about the EU founders using the statement $C_{eu} = Compl((EU, founder, ?f))$.[11] To serialize completeness statements in RDF, we refer the reader to [4]. We now define when a completeness statement is satisfied by an extension pair. To a statement $C = Compl(P_C)$, we associate the `CONSTRUCT` query $Q_C = (\texttt{CONSTRUCT}\ P_C\ P_C)$. Note that, given a graph G, the query Q_C returns the graph consisting of those instantiations of the pattern P_C present in G. For example, the query $Q_{C_{eu}} = (\texttt{CONSTRUCT}\ \{(EU, founder, ?f)\}\ \{(EU, founder, ?f)\})$ returns the founding members of the EU in G. Intuitively, an extension pair (G, G') satisfies a completeness statement C, if the subgraph of G' identified by C is also present in G.

Definition 4 (Satisfaction of Completeness Statements)**.** *An extension pair (G, G') satisfies a completeness statement C, written $(G, G') \models C$, if $[\![Q_C]\!]_{G'} \subseteq G$.*

The above definition naturally extends to the satisfaction of a set $\mathcal{C}$ of completeness statements, that is, $(G, G') \models \mathcal{C}$ iff for all $C \in \mathcal{C}$, it is the case that $[\![Q_C]\!]_{G'} \subseteq G$.

An important tool for characterizing completeness entailment is the transfer operator, which evaluates over G all `CONSTRUCT` queries associated to the statements in $\mathcal{C}$ and returns the union of the results.

Definition 5 (Transfer Operator)**.** *Let $\mathcal{C}$ be a set of completeness statements. The* transfer operator *$T_\mathcal{C}$ maps every graph G to the subgraph*

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

As an illustration, consider the statement C_{eu} as before and the graph $G_{org} = \{(EU, founder, ger), (ASEAN, founder, sgp)\}$. Then, it is the case that $T_{\{C_{eu}\}}(G_{org}) = \{(EU, founder, ger)\}$. We have the following immediate characterization of transfer operator: for all extension pairs (G, G'), it holds that $(G, G') \models \mathcal{C}$ iff $T_\mathcal{C}(G') \subseteq G$.

3. Motivation and Formal Framework

In this section, we motivate and formalize the problem of query completeness and query soundness, adapting the definitions from [4] to the data-aware setting and to the problem of soundness.

3.1. Query Completeness

Given an RDF graph and a set of completeness statements, we want to check whether a query can be answered completely.

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

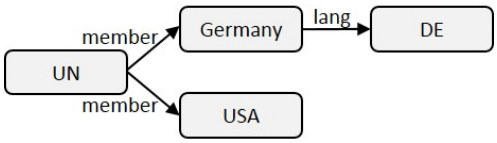

Fig. 2. RDF graph G_{cou} about countries

3.1.1. Motivating Scenario

Consider the RDF graph G_{cou} about members countries of the United Nations (UN) and official languages of the members as in Figure 2. Next, consider the query Q_0 asking for the UN members and their languages:

$$Q_0 = (W_0, P_0) \\ = (\{\, ?m, ?l \,\}, \{(UN, member, ?m), (?m, lang, ?l)\})$$

Evaluating Q_0 over the graph gives only one mapping, where the member is mapped to Germany and the language is mapped to `de`. Up until now, nothing can be said about the completeness of the query since *(i)* there can be another UN member with an official language; *(ii)* Germany may have another language; or *(iii)* the USA may have an official language.

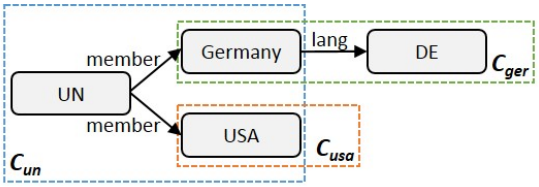

Fig. 3. RDF graph G_{cou} with completeness statements

Let us consider the same graph as above, now enriched with completeness information, as displayed in Figure 3. The figure illustrates the set $\mathcal{C}_{cou}$ of three completeness statements and to which parts of the graph the statements apply:

- $C_{un} = Compl((UN, member, ?m))$, which states that the graph contains all members of the UN;[12]
- $C_{ger} = Compl((ger, lang, ?l))$, which states the graph contains all official languages of Germany;
- $C_{usa} = Compl((usa, lang, ?l))$, which states the graph contains all official languages of the USA (i.e., the USA has no official languages).[13]

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

[12] For the sake of example, let us suppose that this is true.

[13] See, e.g., https://www.cia.gov/library/publications/the-world-factbook/geos/us.html

First, from the statement C_{un} about UN members, we can infer that the part $(UN, member, ?m)$ of Q_0 is complete. By evaluating that part over G_{cou}, we know that all the UN members are Germany and the USA. In terms of extension pairs, that means that no extension $G'_{cou} \supseteq G_{cou}$ satisfying C_{un} has UN members other than Germany and the USA. This allows us to instantiate the query Q_0 to the following two queries that intuitively are together equivalent to Q_0 itself:

- $Q_1 = (W_1, P_1) \\ = (\{\, ?l \,\}, \{(UN, member, ger), (ger, lang, ?l)\})$
- $Q_2 = (W_2, P_2) \\ = (\{\, ?l \,\}, \{(UN, member, usa), (usa, lang, ?l)\}),$

where we record that the variable $?m$ has been instantiated by Germany and the USA, respectively.

Our task is now transformed to checking whether Q_1 and Q_2 can be answered completely. As for Q_2, we know from the statement C_{usa} that our data graph is complete with regard to the triple pattern $(usa, lang, ?l)$. This again allows us to instantiate the query Q_2 wrt. the graph G_{cou}. However, now we come to the situation where there is no matching part in G_{cou}: instantiating the triple pattern $(usa, lang, ?l)$ returns nothing (i.e., the USA has no official languages). In other words, for any possible extension G'_{cou} of G_{cou}, as guaranteed by C_{usa}, the extension G'_{cou} is also empty for the part $(usa, lang, ?l)$. Thus, there is no way that Q_2 will return an answer, so it can be safely removed. Here we can also see that we are complete for Q_2.

Now, only the query Q_1 is left. Again, from the statement C_{ger}, we know that we are complete for the part $(ger, lang, ?l)$ of Q_1. This allows us to instantiate the query Q_1 to the query Q_3, that is intuitively equivalent to Q_1 itself:

$$Q_3 = (W_3, P_3) \\ = (\{\,\}, \{(UN, member, ger), (ger, lang, de)\}),$$

where we record that the variable $?m$ has been instantiated by Germany and $?l$ by `de`. However, our graph is complete for Q_3 as it contains the whole ground body of Q_3. Clearly, no extension G'_{cou} of G_{cou} 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 `de` is the only official language of Germany, the only UN member with an official language.

In summary, 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: *(i)* we find parts of the query that can be guaranteed to be complete by the completeness statements; *(ii)* we produce equivalent query instantiations by evaluating those complete query parts over the graph and applying the obtained mappings to the query itself; *(iii)* 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 completeness reasoning approach of [4], it is not possible to derive the same conclusion. Without looking at the available graph, we cannot conclude that Germany and the USA are all the UN members, since it could be the case that the members are completely different items (i.e., not Germany nor the USA). Consequently, just knowing that the official languages of Germany and the USA are complete does not help in the reasoning.

3.1.2. Formalization of Completeness Reasoning

When querying a data source, we want to know whether the data source provides sufficient information to retrieve all answers to the query, that is, whether the query is *complete* wrt. the real world. For instance, when querying for members of the UN, it would be interesting to know whether we really get *all* such countries. Intuitively, a query is complete over an extension pair whenever all answers we retrieve over the ideal graph are also retrieved over the available graph. We now define query completeness wrt. extension pairs.

Definition 6 (Query Completeness)**.** *To express that a query 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, $[\![Q]\!]_{G'} \subseteq [\![Q]\!]_G$.*[14] *In this case we write $(G, G') \models Compl(Q)$.*

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

Now, the question 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 completeness statements, that is required for computing the query answers, so one can trust the result of the query. In the following, we define completeness entailment.

Definition 7 (Completeness Entailment)**.** *Let $\mathcal{C}$ be a set of completeness statements, G a graph, and Q a query. Then $\mathcal{C}$* and *G* entail the completeness of *Q, written $\mathcal{C}, G \models Compl(Q)$, if for all extension pairs $(G, G') \models \mathcal{C}$, it holds that $(G, G') \models Compl(Q)$.*

In our motivating scenario, we have seen that the graph about the UN and the completeness statements entail the completeness of the query Q_0 asking for members of the UN and their official languages.

Since for our SPARQL queries all variables are distinguished, the set of query answers for such a query is the same as the set of mappings satisfying the BGP. We can therefore focus on the BGPs used in the body of queries for completeness entailment. Note that here (again) the distinction between bag and set semantics collapses. The following proposition provides an initial characterization of completeness entailment, which will serve as a starting point to develop formal notions for completeness checking and an algorithm in Section 4. Basically, for a set of completeness statements, a graph, and a BGP, the completeness entailment holds, if and only if extending the graph with a possible BGP instantiation (by some mapping) such that the extension satisfies the statements, always results in the inclusion of the BGP instantiation in the graph itself.

Proposition 1. *Let $\mathcal{C}$ be a set of completeness statements, G a graph, and P a BGP. Then the following are equivalent:*

1. *$\mathcal{C}, G \models Compl(P)$;*
2. *for every mapping μ such that $dom(\mu) = var(P)$ and $(G, G \cup \mu P) \models \mathcal{C}$, it is the case that $\mu P \subseteq G$.*

In other words, the completeness entailment does not hold, if and only if we can find a possible BGP instantiation (by 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 BGPs 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).

[14] 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.

3.2. Query Soundness

Here, we motivate the second main problem of this work, query soundness. The problem comes in two variants: answer soundness and pattern soundness.

3.2.1. Answer Soundness

In a nutshell, a mapping is a sound answer for a query with negation over a given graph if it continues to be an answer over all possible completions of the graph. For an example, 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:

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

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

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

For this graph, consider also the set $\mathcal{C}_l$ of the following four completeness statements: the first two are C_{ger} and C_{usa} as we have had before in the motivating scenario of query completeness. The other two are C_{spa} for all official languages of Spain and C_{eu} for all EU founders.[15] Note that we do not claim anything about the completeness of the official languages of Singapore (= sgp).

Evaluating the graph pattern over the graph in the standard way results in $[\![P_l]\!]_{G_l} = \{\{?c \mapsto usa\}, \{?c \mapsto sgp\}, \{?c \mapsto spa\}\}$. 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. $\mathcal{C}_l$ over which the answers are not returned.

Let us analyze $\{?c \mapsto usa\}$. First, we check if $(usa, lang, en)$ is certainly not true. Indeed, since we know by the graph and the statement C_{usa} that the USA has no official languages, the triple $(usa, lang, en)$ must not be true. Second, we check if $\{(usa, lang, ?l), (?f, lang, ?l), (EU, founder, ?f)\}$ 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 $\{?c \mapsto usa\}$ is sound.

Next, let us analyze $\{?c \mapsto sgp\}$. We check if $(sgp, lang, en)$ is indeed not true, that is, there is no valid extension where $(sgp, lang, en)$ 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 $\{?c \mapsto sgp\}$.

Last, let us analyze $\{?c \mapsto spa\}$. First, we check if the triple $(spa, lang, en)$ is not true. Since we know by C_{spa} and the graph that Spain's official language is only `es`, then $(spa, lang, en)$ must not be true. Second, we check if the BGP $\{(spa, lang, ?l), (?f, lang, ?l), (EU, founder, ?f)\}$ 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 therefore conclude that the answer $\{?c \mapsto spa\}$ is sound.

In summary, our analysis established for which answers the `NOT-EXISTS` patterns of the query pattern are surely false and thus whether the answer is sound.

3.2.2. Pattern Soundness

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

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

Consider also the set $\mathcal{C}_f$ consisting of two completeness statements: C_{lang} for all languages of countries and C_{eu} for all EU founders. We will show that the statements guarantee that all answers returned by P_f are sound, independently of the queried graph. In such a case, we say that the pattern itself is *sound*.

Let us see why $\mathcal{C}_f$ guarantees for any possible graph the soundness of all answers to P_f. Consider a graph G and suppose that pattern evaluation over G returns $\{?c \mapsto \bar{c}\}$ for an IRI $\bar{c}$. Consider also an arbitrary extension G' of G such that $(G, G') \models \mathcal{C}_f$. To show that $\{?c \mapsto \bar{c}\}$ is sound, we must make sure that neither over G nor over G' does $\bar{c}$ have `en` as an official language and is $\bar{c}$ 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 lan-

[15] For the sake of example, suppose this is true.

guages of $\bar{c}$. The fact that $\bar{c}$ is returned by P_f over G means that `en` is not among its official languages according to G, and due to completeness, also not according to G'. Moreover, the fact that $\bar{c}$ is returned over G means that $\bar{c}$ is not an EU founder according to G, and, since G is complete for all EU founders due to C_{eu}, also not according to G'. Thus we can be sure that the answer $\{?c \mapsto \bar{c}\}$ is sound. Since the answer and the graph were arbitrary, we conclude that the set $\mathcal{C}_f$ of completeness statements entails the soundness of P_f.

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

3.3. Formalization of Soundness Reasoning

In the following, we provide definitions of answer soundness and pattern soundness.

Definition 8 (Answer Soundness)**.** *To express that a mapping μ is sound for a graph pattern P, we write $Sound(\mu, P)$. We say that an extension pair (G, G') satisfies $Sound(\mu, P)$ if, whenever $\mu \in [\![P]\!]_G$, then also $\mu \in [\![P]\!]_{G'}$. In this case we write $(G, G') \models Sound(\mu, P)$.*

From the above definition, for $\mu \notin [\![P]\!]_G$ it is trivial that $(G, G') \models Sound(\mu, P)$. Thus, we are only interested in the soundness of answers occurring in $[\![P]\!]_G$.

Definition 9 (Answer Soundness Entailment)**.** *Given a set $\mathcal{C}$ of completeness statements, a graph G, a graph pattern P, and a mapping $\mu \in [\![P]\!]_G$, we say that $\mathcal{C}$ and G entail the soundness of μ for P, written as $\mathcal{C}, G \models Sound(\mu, P)$, if for all extension pairs $(G, G') \models \mathcal{C}$, it holds that $(G, G') \models Sound(\mu, P)$.*

In our motivating scenario we saw that for all possible completions of the graph, *usa* is a sound answer while *sgp* is not. Thus, $\mathcal{C}_l, G_l \models Sound(\{?c \mapsto usa\}, P_l)$, whereas $\mathcal{C}_l, G_l \not\models Sound(\{?c \mapsto sgp\}, P_l)$.

Pattern soundness, as opposed to answer soundness, is concerned with a graph pattern as a whole and abstracts from any specific answers of the pattern.

Definition 10 (Pattern Soundness)**.** *We express that a graph pattern P is* sound *by $Sound(P)$. We say that P* is sound over the extension pair (G, G')*, written $(G, G') \models Sound(P)$, if $[\![P]\!]_G \subseteq [\![P]\!]_{G'}$.*

Thus, a pattern is sound whenever *all* answers in the evaluation over G are also present in that over G'. Entailment of pattern soundness by completeness statements is defined in a natural manner.

Definition 11 (Pattern Soundness Entailment)**.** *A set of completeness statements $\mathcal{C}$* entails *the soundness of a graph pattern P, written $\mathcal{C} \models Sound(P)$, if for all extension pairs $(G, G') \models \mathcal{C}$, it holds that $(G, G') \models Sound(P)$.*

In our motivating scenario, it is the case that $\mathcal{C}_f \models Sound(P_f)$. Combining the definitions above, we see that a pattern is sound if and only if over all graphs all its answers are sound.

Proposition 2. *Let $\mathcal{C}$ be a set of completeness statements and P be a graph pattern. Then, $\mathcal{C} \models Sound(P)$ iff $\mathcal{C}, G \models Sound(\mu, P)$ for every graph G and mapping $\mu \in [\![P]\!]_G$.*

4. Checking Query Completeness

In this section, we introduce formal notions and present an algorithm for checking the entailment of query completeness. We also analyze the complexity of the entailment problem.

4.1. Formal Notions

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. We define a *partially mapped BGP* as a pair (P, μ) where P is a BGP and μ is a mapping with $dom(\mu) \cap var(P) = \emptyset$. 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 $P \equiv (P, \mu_\emptyset)$. Furthermore, we define the evaluation of a set of partially mapped BGPs over a graph G as the union of the evaluations of each of them over G.

Example 3. Consider our motivating scenario. Over the BGP P_0 of the query Q_0, instantiating the variable $?m$ to *ger* results in the BGP P_1 of the query Q_1. Pairing P_1 with this instantiation gives the partially mapped BGP $(P_1, \{\, ?m \mapsto ger \,\})$. Moreover, it is the case that $[\![(P_1, \{\, ?m \mapsto ger \,\})]\!]_{G_{cou}} = \{\, \{\, ?m \mapsto ger, ?l \mapsto de \,\} \,\}$.

Next, we formalize when two partially mapped BGPs are equivalent wrt. a set $\mathcal{C}$ of completeness statements and a graph G. Essentially, this is the case if they return the same answers over all possible extensions of G. We need this notion to capture the equivalence of the BGP instantiations that resulted from the evaluation of complete BGP parts.

Definition 12 (Equivalence under $\mathcal{C}$ and G)**.** *Let (P,μ) and (P',ν) be partially mapped BGPs, $\mathcal{C}$ be a set of completeness statements, and G be a graph. Then (P,μ) is* equivalent *to (P',ν) wrt. $\mathcal{C}$ and G, written $(P,\mu) \equiv_{\mathcal{C},G} (P',\nu)$, if for all $(G,G') \models \mathcal{C}$, it holds that $[\![(P,\mu)]\!]_{G'} = [\![(P',\nu)]\!]_{G'}$.*

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

Example 4. Consider the queries in our motivating scenario in Section 3.1.1. The equivalences discussed there can now be stated as $\{ (P_0,\mu_\emptyset) \} \equiv_{\mathcal{C}_{cou},G_{cou}} \{ (P_1,\{ ?m \mapsto ger \}),(P_2,\{ ?m \mapsto usa \}) \} \equiv_{\mathcal{C}_{cou},G_{cou}} \{ (P_3,\{ ?m \mapsto ger, ?l \mapsto de \}) \}$.

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, and the remaining parts are matched to the graph. For this reason, using the transfer operator from Definition 5, we define

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

as the *crucial part of P wrt. $\mathcal{C}$ and G*. In the crucial part computation, we evaluate the $T_{\mathcal{C}}$ operator over the union of the prototypical graph $\tilde{P}$ and the graph G in order to obtain the complete parts wrt. the set $\mathcal{C}$ of completeness statements. However, we are only interested in the complete parts that are overlapping with our BGP P. Therefore, we undo the 'freezing' effect of the prototypical graph by the melt operator $\tilde{id}^{-1}$ to transform it back into a BGP, and then compute the intersection with the BGP P to get the complete parts that are relevant to P.[16]

By its construction, we are complete for the crucial part, that is, $\mathcal{C},G \models Compl(cruc_{\mathcal{C},G}(P))$. Later on, we will see that the crucial part can be used to guide the instantiation process during completeness entailment checking.

Example 5. Consider the query $Q_0 = (W_0,P_0)$ in our motivating scenario. Applying the freeze mapping $\tilde{id} = \{ ?m \mapsto \tilde{m}, ?l \mapsto \tilde{l} \}$ to P_0, we obtain $\tilde{P}_0 = \{(UN,member,\tilde{m}),(\tilde{m},lang,\tilde{l})\}$. Among the three completeness statements in $\mathcal{C}_{cou}$, only C_{un} and C_{ger} can be applied to $\tilde{P}_0 \cup G_{cou}$ and their application results in $T_{\mathcal{C}_{cou}}(\tilde{P}_0 \cup G_{cou}) = \{(UN,member,usa),(UN,member,ger),(UN,member,\tilde{m}),(ger,lang,de)\}$. Undoing the freezing and intersecting with P_0 results in $cruc_{\mathcal{C}_{cou},G_{cou}}(P_0) = P_0 \cap \tilde{id}^{-1}(T_{\mathcal{C}_{cou}}(\tilde{P}_0 \cup G_{cou})) = \{ (UN,member,?m) \}$. Consequently, the graph G_{cou} provides us with a complete instantiation of the UN members.

Consider next the pattern $P_2 = \{(UN,member,usa),(usa,lang,?l)\}$. Now, all completeness statements in $\mathcal{C}_{cou}$ apply to the frozen version of P_2 and the graph, which gives us $T_{\mathcal{C}_{cou}}(\tilde{P}_2 \cup G_{cou}) = \{(UN,member,ger),(UN,member,usa),(ger,lang,de),(usa,lang,\tilde{l})\}$. Melting the frozen variable and intersecting with P_2 then results in $cruc_{\mathcal{C}_{cou},G_{cou}}(P_2) = P_2$.

Consider now the pattern $P_3 = \{(UN,member,ger),(ger,lang,de)\}$. Then P_3 is ground and $P_3 = \tilde{P}_3 \subseteq G_{cou}$. Only the statements C_{un} and C_{ger} apply to G_{cou}, yielding $T_{\mathcal{C}_{cou}}(\tilde{P}_3 \cup G_{cou}) = T_{\mathcal{C}_{cou}}(G_{cou}) = G_{cou}$. Thus, $cruc_{\mathcal{C}_{cou},G_{cou}}(P_3) = P_3 \cap G_{cou} = P_3$.

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

Definition 13 (Equivalent Partial Grounding)**.** *Let $\mathcal{C}$ 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:

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

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. It holds basically since the instantiation is done over the crucial part, which is complete wrt. $\mathcal{C}$ and G.

Proposition 3 (Equivalent Partial Grounding)**.** *Let $\mathcal{C}$ be a set of completeness statements, G a graph, and (P,ν) a partially mapped BGP. Then*

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

Example 6. Consider our motivating scenario. Recall from Example 5 the crucial parts of the BGPs P_2, P_3, and P_0 from our motivating example:

- $cruc_{\mathcal{C}_{cou},G_{cou}}(P_2) = P_2$;

[16] Here we slightly abuse the notation by performing a mapping (which itself is actually a 'reverse' mapping) over a graph instead of a BGP.

- $cruc_{\mathcal{C}_{cou},G_{cou}}(P_3) = P_3$;
- $cruc_{\mathcal{C}_{cou},G_{cou}}(P_0) = P_{un} = \{(UN, member, ?m)\}$.

Evaluating the crucial parts over the graph G_{cou} yields

- $[\![P_2]\!]_{G_{cou}} = \emptyset$;
- $[\![P_3]\!]_{G_{cou}} = \{\mu_\emptyset\}$;
- $[\![P_{un}]\!]_{G_{cou}} = \{\{?m \mapsto ger\}, \{?m \mapsto usa\}\}$.

The corresponding equivalent partial groundings are

- $epg((P_2, \{?m \mapsto usa\}), \mathcal{C}_{cou}, G_{cou}) = \emptyset$;
- $epg((P_3, \{?m \mapsto ger, ?l \mapsto de\}), \mathcal{C}_{cou}, G_{cou}) = \{(P_3, \{?m \mapsto ger, ?l \mapsto de\})\}$;
- $epg((P_0, \mu_\emptyset), \mathcal{C}_{cou}, G_{cou}) = \{(P_1, \{?m \mapsto ger\}), (P_2, \{?m \mapsto usa\})\}$.

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

- If $[\![cruc_{\mathcal{C},G}(P)]\!]_G = \emptyset$, it returns an empty set.
- If $[\![cruc_{\mathcal{C},G}(P)]\!]_G = \{\mu_\emptyset\}$, it returns $\{(P, \nu)\}$.
- Otherwise, it returns a non-empty set of partially mapped BGPs, where some variables in P are instantiated.

Let us describe what these three cases mean to our completeness entailment procedure, and how their applications lead to termination. 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 USA's official languages case). As demonstrated in Example 6, the first case removes partially mapped BGPs. This is due to the emptiness of the crucial parts. Now, for the third case, it corresponds to the instantiation of complete parts (that is, the crucial parts) of the BGP. The third case generates more specific partially mapped BGPs that are equivalent to the original partially mapped BGP.

As for the second case, it generates exactly the same partially mapped BGP as the original one. Here, we need a special treatment. We first define that a partially mapped BGP (P, ν) is *saturated* wrt. $\mathcal{C}$ and G, if $epg((P, \nu), \mathcal{C}, G) = \{(P, \nu)\}$, that is, if the second case above applies. Note that the notion of saturation is independent of the mapping in a partially mapped BGP: given a mapping ν, a partially mapped BGP (P, ν) is saturated wrt. $\mathcal{C}$ and G iff (P, ν') is saturated wrt. $\mathcal{C}$ and G for any mapping ν'. Thus, wrt. $\mathcal{C}$ and G we say that a BGP P is saturated if $(P, \mu_\emptyset)$ is saturated.

Saturated BGPs hold the key as to whether our completeness entailment check succeeds or not: completeness of saturated BGPs is simply checked by testing whether they are contained in the graph G. Furthermore, once the repeated *epg* applications with the above three cases hit the saturated case (i.e., the second case), we can readily check completeness entailment.[17] Note that this ensures also the termination of the *epg* applications.

Lemma 1 (Completeness Entailment of Saturated BGPs)**.** *Let P be a BGP, $\mathcal{C}$ a set of completeness statements, and G a graph. Suppose P is saturated wrt. $\mathcal{C}$ and G. Then:*

$$\mathcal{C}, G \models Compl(P) \quad \textit{iff} \quad P \subseteq G.$$

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

4.2. Algorithm

From the above notions, we have defined the *cruc* operator 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. Algorithm 1 computes a function that, given a set of completeness statements $\mathcal{C}$, a graph G, and a BGP P, returns the set

$$sat(P, \mathcal{C}, G)$$

of all mappings that have two properties: each BGP instantiation of the mappings constitutes a saturated BGP wrt. $\mathcal{C}$ and G; and the original BGP is equivalent wrt. $\mathcal{C}$ and G with the BGP instantiations produced from all the resulting mappings of the algorithm.

Let us now describe how Algorithm 1 works. Consider a BGP P_{orig}, a set $\mathcal{C}$ of completeness statements, and a graph G. First, we transform our original BGP P_{orig} into its equivalent partially mapped BGP $(P_{orig}, \mu_\emptyset)$ 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 $epg((P, \nu), \mathcal{C}, G)$.

[17] Essentially, it might be that the *epg* applications hit the first case only, that is, there are no more partially mapped BGPs that have to be checked for completeness. For this case, completeness entailment trivially holds.

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

Input: A BGP P_{orig}, a set $\mathcal{C}$ of completeness statements, a graph G
Output: A set Ω of mappings

```
1  P_working ← { (P_orig, μ_∅) }
2  Ω ← ∅
3  while P_working ≠ ∅ do
4      (P, ν) ← takeOne(P_working)
5      P_equiv ← epg((P, ν), C, G)
6      if P_equiv = { (P, ν) } then
7          Ω ← Ω ∪ { ν }
8      else
9          P_working ← P_working ∪ P_equiv
10     end
11 end
12 return Ω
```

As discussed above there can be three result cases here: *(i)* If $epg((P,\nu),\mathcal{C},G) = \emptyset$, then we simply remove (P,ν) and will not consider it anymore in the later iteration; *(ii)* If $epg((P,\nu),\mathcal{C},G) = \{(P,\nu)\}$, that is, (P,ν) is saturated, then we add 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 next proposition about the function computed by Algorithm 1 follows from the construction of the algorithm and Proposition 3.

Proposition 4. *Let P be a BGP, $\mathcal{C}$ a set of completeness statements, and G a graph. Then the following properties hold:*

- $\{(P,\mu_\emptyset)\} \equiv_{\mathcal{C},G} \{(\mu P,\mu) \mid \mu \in sat(P,\mathcal{C},G)\}$;
- *μP is saturated wrt. $\mathcal{C}$ and G, for all mappings $\mu \in sat(P,\mathcal{C},G)$.*

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

Theorem 1 (Completeness Entailment Check)**.** *Let P be a BGP, $\mathcal{C}$ a set of completeness statements, and G a graph. Then the following are equivalent:*

1. $\mathcal{C}, G \models Compl(P)$;
2. $\mu P \subseteq G$, *for all* $\mu \in sat(P,\mathcal{C},G)$.

Example 7. Consider our motivating scenario. Then $sat(P_0,\mathcal{C}_{cou},G_{cou}) = \{\{?m \mapsto ger, ?l \mapsto de\}\}$. For every mapping μ in $sat(P_0,\mathcal{C}_{cou},G_{cou})$, it holds that $\mu P_0 \subseteq G_{cou}$. Thus, by Theorem 1 the entailment $\mathcal{C}_{cou}, G_{cou} \models Compl(P_0)$ holds.

From looking back at the initial characterization of completeness entailment in Proposition 1, 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 1 it is sufficient for completeness entailment checking to consider only the mappings in $sat(P,\mathcal{C},G)$, which we know how to compute.

4.2.1. Simple Practical Considerations

Following from the aforementioned algorithm to compute saturated mappings, and Theorem 1, a corresponding algorithm for checking completeness entailment can be easily derived. In what follows we provide two simple heuristic techniques of the completeness checking algorithm: early failure detection and completeness skip. More elaborate heuristics are given in Section 6.

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 1). 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.

Completeness Skip. Recall the definition of *epg* as $epg((P,\nu),\mathcal{C},G) = \{(\mu P, \nu \cup \mu) \mid \mu \in [\![cruc_{\mathcal{C},G}(P)]\!]_G\}$, which relies on the *cruc* operator. Now, suppose that $cruc_{\mathcal{C},G}(P) = P$, 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. $\mathcal{C}$ and G. In conclusion, whenever $cruc_{\mathcal{C},G}(P) = P$, we just remove the corresponding (P,ν) from $\mathbf{P}_{working}$ and thus skip its instantiations.

4.3. Complexity

In this subsection, we analyze the complexity of the problem of data-aware completeness entailment. While the complexity of checking data-agnostic completeness entailment is NP-complete [4], the addition of the data graph to the entailment increases the complexity, which is now Π_2^P-complete. The hardness is shown by a reduction of the validity problem for $\forall\exists$3SAT formulas.

Proposition 5. *Deciding the entailment $\mathcal{C}, G \models Compl(P)$, given a set $\mathcal{C}$ of completeness statements, a graph G, and a BGP P, is Π_2^P-complete.*

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.

The following proposition states that the entailment does not become easier if the graph is fixed. The reason is that the reduction from the validity problem of a $\forall\exists$3SAT formula uses one fixed graph.

Proposition 6. *Let G be a graph. Deciding the entailment $\mathcal{C}, G \models Compl(P)$, given a set $\mathcal{C}$ of completeness statements and a BGP P, is in Π_2^P. There is a graph for which the problem is Π_2^P-complete.*

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 7. *Let P be a BGP. Deciding the entailment $\mathcal{C}, G \models Compl(P)$, given a set $\mathcal{C}$ of completeness statements and a graph G, is in NP. There is a BGP for which the problem is NP-complete. This still holds when the graph is fixed.*

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

Proposition 8. *Let $\mathcal{C}$ be a set of completeness statements. Deciding the entailment $\mathcal{C}, G \models Compl(P)$, given a graph G and a BGP P, is in CoNP. There is a set $\mathcal{C}$ of completeness statements for which the problem is CoNP-complete. This still holds when the graph is fixed.*

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

Proposition 9. *Let $\mathcal{C}$ be a set of completeness statements and P be a BGP. Deciding the entailment $\mathcal{C}, G \models Compl(P)$, given a graph G, is in PTIME.*

This result corresponds to some practical cases when queries are assumed to be of limited length[18] and hence, so are completeness statements (which are essentially also queries).

Table 1
Complexity table for the data-aware completeness entailment problem with various inputs fixed ('×' denotes 'fixed')

input			*complexity*
$\mathcal{C}$	G	P	
✓	✓	✓	Π_2^P-complete
✓	×	✓	Π_2^P-complete
✓	✓	×	NP-complete
✓	×	×	NP-complete
×	✓	✓	CoNP-complete
×	×	✓	CoNP-complete
×	✓	×	in PTIME

Our complexity results with various inputs fixed are summarized in Table 1. From this complexity study, it is therefore of our interest to investigate how well the problem of completeness entailment may be solved in practice. In later sections, we will provide heuristic techniques, as well as experimental evaluations of the problem.

Completeness of Queries with Projections The above formalization deals with the completeness of queries with no projections (that is, where all variables in the BGP are distinguished). One may wonder, what happens when not all variables are distinguished? The answer depends on whether bag or set semantics is used.

With bag semantics, due to the monotonicity of BGPs, and answer-multiplicity being preserved, our results immediately transfer to queries with projections: a query with projections is complete if and only if the projection-free version of the query is complete.

We do not know a characterization of completeness for queries with projections under set semantics. Nevertheless, Theorem 1 derives a sufficient condition for completeness in this case: whenever the BGP of a query with projections is complete, then the query is also complete under set semantics.

5. Checking Query Soundness

In this section, we leverage completeness reasoning for checking answer and pattern soundness.

[18] as also customary in database theory when analyzing the data complexity of query evaluation [13]

5.1. Checking Answer Soundness

We will use data-aware completeness reasoning to judge whether an answer obtained by evaluating a graph pattern over a graph is sound.

Example 8. Remember the motivating scenario of answer soundness in Section 3.2.1. Consider the mapping $\{?c \mapsto usa\} \in [\![P_l]\!]_{G_l}$. After instantiating the variable $?c$ in the two negated subpatterns with usa, we obtain the patterns $(usa, lang, en)$ and $(usa, lang, ?l), (?f, lang, ?l), (EU, founder, ?f)$. Using the techniques for completeness checking in Section 4, we find that both the entailment $\mathcal{C}_l, G_l \models Compl((usa, lang, en))$ and the entailment

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

hold. Therefore, these subpatterns will fail over every extension of G_l compatible with $\mathcal{C}_l$, and $\{?c \mapsto usa\}$ will continue to be an answer, that is formally, $\mathcal{C}_l, G_l \models Sound(\{?c \mapsto usa\}, P_l)$.

In contrast, consider the mapping $\{?c \mapsto sgp\} \in [\![P_l]\!]_{G_l}$. For the extension $G_l' = G_l \cup \{(sgp, lang, en)\}$, however, we have $\{?c \mapsto sgp\} \notin [\![P_l]\!]_{G_l'}$, since instantiating the negated subpattern $(?c, lang, en)$ to $(sgp, lang, en)$ results in a triple satisfied by G_l'. Clearly, $(G_l, G_l') \models \mathcal{C}_l$, from which we conclude that $\mathcal{C}_l, G_l \not\models Compl((sgp, lang, en))$. In short, $\{?c \mapsto sgp\}$ is not a sound answer for P_l over G_l, because $\mathcal{C}_l$ and G_l do not entail the completeness of the instantiated triple pattern $\{?c \mapsto sgp\}(?c, lang, en)$.

The main theorem of this subsection generalizes the observations of the example. Intuitively, it states that the soundness of some answer-mapping of a graph pattern over a graph is guaranteed exactly if all the graph pattern's NOT-EXISTS-BGPs, after applying the answer-mapping to them, are complete for the graph.

Theorem 2. (ANSWER SOUNDNESS) *Let G be a graph, $\mathcal{C}$ a set of completeness statements, P a graph pattern, and $\mu \in [\![P]\!]_G$ a mapping. Then the following are equivalent:*

1. $\mathcal{C}, G \models Sound(\mu, P)$;
2. $\mathcal{C}, G \models Compl(\mu P_i)$, *for all* $P_i \in P^-$.

In fact, Theorem 2 holds for a wider class of graph patterns than defined in this article. We only need the positive part of the pattern to 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 2, the check of 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 Section 4). 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 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 7 reports an experimental study of answer soundness checking in practical settings.

5.2. Checking Pattern Soundness

As shown 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 $Compl(P \mid P')$. Given an extension pair (G, G'), we define that $(G, G') \models Compl(P \mid P')$ if $[\![(var(P), P \cup P')]\!]_{G'} \subseteq [\![P]\!]_G$.[19] 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 $Compl((?c, lang, en) \mid (?c, a, country))$ denotes the completeness of all things having English as their language, provided that those things are of type country. Note 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 $\mathcal{C} \models Compl(P \mid P')$ holds if for all extension pairs (G, G') satisfying $\mathcal{C}$, it is

[19] Though redundancies may occur in the left-hand side due to the projection, for our characterizations here the $\subseteq$ operation is done under set semantics. Hence, the redundancies are ignored.

the case that $(G, G') \models Compl(P \mid P')$. The following proposition states that such an entailment holds iff the application of the transfer operator $T_{\mathcal{C}}$ to the prototypical graph $\tilde{P} \cup \tilde{P}'$ contains $\tilde{P}$. Recall that the prototypical graph represents any possible graph that satisfies a BGP.

Proposition 10. *For a set $\mathcal{C}$ of completeness statements and BGPs P and P', it is the case that*

$$\mathcal{C} \models Compl(P \mid P') \quad \textit{iff} \quad \tilde{P} \subseteq T_{\mathcal{C}}(\tilde{P} \cup \tilde{P}').$$

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

$$\{(\tilde{c}, lang, en)\} \subseteq \{(\tilde{c}, lang, en)(\tilde{c}, a, country)\} = T_{\mathcal{C}_f}(\{(\tilde{c}, lang, en), (\tilde{c}, a, country)\})\}.$$

This means that the set $\mathcal{C}_f$ 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 2. *Let $\mathcal{C}$ be a set of completeness statements and P a graph pattern. Then $\mathcal{C} \models Sound(P)$ provided that $\mathcal{C} \models Compl(P_i \mid P^+)$ for all $P_i \in P^-$.*

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

Example 9. Consider the following graph patterns:

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

$$P_2 = \{(?c, a, country), \neg\exists\{(?c, lang, en), (?c, lang, ?l)\}\}.$$

Suppose both return as answers mappings $\{?c \mapsto \bar{c}\}$ for an IRI $\bar{c}$. It is the case that $\bar{c}$ is a country that does not have the language `en`. The reason is that, after instantiating $?c$ with $\bar{c}$, all three negated subpatterns are empty over a graph G if and only if the triple $(\bar{c}, lang, en)$ is not present in G. Consider next also the completeness statement $\mathcal{C} = \{Compl((?c, lang, en))\}$. If the triple $(\bar{c}, lang, en)$ is not present in G, and $(G, G') \models \mathcal{C}$, then the triple is not present in G' either. Hence, all three negated subpatterns fail and $\{?c \mapsto \bar{c}\}$ persists to be an answer to P_1 and P_2 over G'. We thus have $\mathcal{C} \models Sound(P_1)$ and $\mathcal{C} \models Sound(P_2)$ despite the violation of the right-hand side of Lemma 2.

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 $(?c, lang, ?l)$ is superfluous since the emptiness of the BGP of the `NOT-EXISTS` pattern only depends on the triple pattern $(?c, lang, en)$. Consequently, for both cases having only the completeness statement $Compl((?c, lang, en))$ is sufficient to guarantee their soundness.

To identify such redundancies, we associate to each pattern $P_i \in P^-$ the query $(var(P^+), P^+ \cup P_i)$. These are essentially conjunctive queries, which can be checked for set containment and which can be minimized (see [14]). Specifically, we propose a normal form for graph patterns, called *non-redundant form (NRF)*. A graph pattern P is in NRF if it satisfies two conditions: there are no redundant negated patterns, and there are no non-minimal negated patterns. This can be formalized as follows:

- The BGP $P_i \in P^-$ is *redundant* if there is a distinct $P_j \in P^-$ such that

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

- The BGP $P_i \in P^-$ is *non-minimal* if there is a non-empty strict subpattern $P_i' \subset P_i$ such that

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

With this notion in place, we can obtain the main theorem of this subsection. The theorem states that given an NRF graph pattern, the check of whether it is sound can be reduced to checking whether each BGP among the `NOT-EXISTS` patterns is complete under the condition of the positive part. Thus, the theorem ensures that the converse of Lemma 2 holds for NRF graph patterns.

Theorem 3. (PATTERN SOUNDNESS) *Let $\mathcal{C}$ be a set of completeness statements and P a graph pattern in NRF. Then the following are equivalent:*

1. $\mathcal{C} \models Sound(P)$;
2. $\mathcal{C} \models Compl(P_i \mid P^+)$ *for all* $P_i \in P^-$.

Example 10. In the motivating scenario of pattern soundness, it holds that

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

By Theorem 3, it is the case $\mathcal{C}_f \models \mathit{Sound}(P_f)$.

Complexity If P is a graph pattern (which may have negated parts) and $\bar{P}$ is obtained from P by eliminating redundant negated patterns and minimizing the remaining ones, we say that $\bar{P}$ is an NRF of P. Clearly, P and every such $\bar{P}$ are equivalent. According to Theorem 3, a straightforward attempt at implementing a soundness check for patterns P would be to compute an NRF $\bar{P}$ of P and then perform the completeness entailment checks specified by the theorem. Such a $\bar{P}$ can be computed by a function that runs in polynomial time and makes several calls to an NP-oracle, while the entailment checks can be performed in NP. This gives us P^{NP} as an upper complexity bound for checking pattern soundness. We will show that the problem is even in NP.

For that purpose we introduce a relationship between patterns with negation of which the relationship between a pattern and its NRF is a special case. We say that two graph patterns P, $\bar{P}$ are *negation-similar* if the following holds:

- $P^+ = \bar{P}^+$;
- for every $P_i \in P^-$, there is a $\bar{P}_j \in \bar{P}^-$ such that $(\mathit{var}(P^+), P^+ \cup P_i) \sqsubseteq_s (\mathit{var}(\bar{P}^+), \bar{P}^+ \cup \bar{P}_j)$;
- for every $\bar{P}_j \in \bar{P}^-$, there is a $P_i \in P^-$ such that $(\mathit{var}(\bar{P}^+), \bar{P}^+ \cup \bar{P}_j) \sqsubseteq_s (\mathit{var}(P^+), P^+ \cup P_i)$.

Note that negation-similarity is in NP, since it can be checked by guessing for every P_i a corresponding $\bar{P}_j$, for every $\bar{P}'_j$ a corresponding P'_i and then performing containment checks by guessing and verifying query homomorphisms (see [14]).

Negation-similarity is a key-property here because it preserves equivalence.

Proposition 11. *Negation-similar graph patterns are equivalent.*

With this background we can design an NP soundness check for patterns P by, intuitively, first guessing $\bar{P}$, verifying that it is negation-similar to P, and running the completeness entailment checks of the theorem. We first make the step of guessing $\bar{P}$ more formal.

Proposition 12. *Let P be a graph pattern. Then there exists a graph pattern $\bar{P}$ such that*

- *$\bar{P}$ is in NRF;*
- *$\bar{P}$ is obtained from P by dropping negated patterns and by dropping atoms from negated patterns;*
- *$\bar{P}$ is similar to P.*

We can obtain such a $\bar{P}$ as an NRF of P by first dropping redundant negated patterns until there are no redundancies any more and then dropping atoms from the remaining $P_i \in P^-$ during the minimization of the $(\mathit{var}(P^+), P^+ \cup P_i)$. Clearly, the redundancy removal and minimization steps preserve negation-similarity.

Now we have all ingredients in place for an NP-algorithm that checks whether a set of completeness statements $\mathcal{C}$ entails the soundness of a graph pattern P:

(i) nondeterministically drop negated patterns from P;
(ii) nondeterministically drop atoms from the remaining negated patterns;
(iii) verify that the resulting pattern $\bar{P}$ is similar to P;
(iv) verify that $\mathcal{C} \models \mathit{Compl}(\bar{P}_j \mid \bar{P}^+)$ for every $\bar{P}_j \in \bar{P}^-$.

Clearly, the first two steps can be performed in nondeterministic polynomial time. As seen above, also, negation-similarity can be checked in nondeterministic polynomial time. Finally, by Proposition 10, the completeness checks in Step (iv) can be reduced to a linear number of T_C applications, which are basically evaluations of conjunctive `CONSTRUCT` queries and are therefore in NP. Thus, all the checks together can be performed in NP.

If the resulting $\bar{P}$ passes the completeness checks, then we have found a query that is sound by Lemma 2 and is equivalent to P by Proposition 11 so that P is sound. If P is sound, then any NRF of P is a pattern that can be guessed and verified by the above algorithm. This shows membership in NP.

NP-hardness is clear, since Theorem 3 also shows that completeness checking can be reduced to soundness checking.

Proposition 13. *Soundness checking of graph patterns is NP-complete.*

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. Section 7 reports an experimental investigation of pattern soundness checking in practical cases.

Soundness of Queries with Projections We have provided a full characterization of the soundness of queries with negation where no projections are involved (or where the projection is over all variables in the query body). One may wonder whether our characterization here can also be used for queries with negation that involve generic projections (that is, where some variables can be non-distinguished). The next example shows that in general, the condition from Theorem 3 is not a necessary condition for pattern soundness entailment of queries with projection.

Example 11. Consider the following boolean query, which asks whether there is some triple that cannot be right-shifted.

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

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

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

If we assume bag semantics, Q returns a bag of empty mappings $\mu_\emptyset$ when evaluated over a graph G, one for each answer to the graph pattern of the query. We show that if G' is such that $(G, G') \models \mathcal{C}$, then the number of empty mappings returned by Q over G' is no less than the number retrieved over G (as otherwise, it will be not sound). Now suppose that Q returns a copy of $\mu_\emptyset$ over G. Then there is a triple $(a, b, c) \in G$ such that $(c, a, b) \notin G$. If also $(c, a, b) \notin G'$, then Q over G' continues to return a copy of $\mu_\emptyset$ for the same reason as before. Consider therefore the case that $(c, a, b) \in G'$. If $(b, c, a) \notin G'$, then Q returns again a copy of $\mu_\emptyset$, this time because the triple $(c, a, b) \in G'$ does not have its right shift in G'. If, however, also $(b, c, a) \in G'$, then the completeness statement kicks in and enforces that $(c, a, b) \in G$, which contradicts our original assumption. Thus, $\mathcal{C}$ entails the pattern soundness of Q, even though $\mathcal{C}$ does not entail $Compl((?z, ?x, ?y) \mid (?x, ?y, ?z))$.

We do not know a characterization of pattern soundness for queries involving projections. Nevertheless, Theorem 3 still gives a sufficient condition for soundness in this case.

Combining Soundness and Completeness Reasoning A graph pattern with negation can be both sound and complete. Theorem 3 characterizes when a graph pattern P in NRF is sound wrt. a set $\mathcal{C}$ of completeness statements. One can show that P is complete if and only if the positive part P^+ is complete. For the right-to-left direction, it can be seen that whenever the positive part is complete, if there is an extension to the graph G, the whole graph pattern P can only remove mappings (that is, it cannot add new mappings). Hence, the evaluation result of P over G is guaranteed to contain that over G'. For the left-to-right direction, whenever the positive part cannot be guaranteed to be complete wrt. the set C of completeness statements, we can build a counterexample extension pair of the pattern soundness entailment by putting the freeze version of the positive part as the graph G', and the T_C application over G' as the graph G. Via both characterizations, we can then check whether a graph pattern is sound and/or complete.

6. Heuristics for Completeness Checking

So far, we have formally characterized completeness entailment, and, for soundness entailment, provided a reduction to completeness entailment. In this section we investigate how completeness checks can actually be implemented in practice. A vanilla implementation of completeness reasoning over a Wikidata-based experiment setting took about 15 s on average for a single completeness check, whereas query evaluation took just about 1 ms. Such a considerable overhead over query evaluation may hinder the adoption of our completeness framework in practical settings. This motivates the need for heuristic techniques that may provide speed-ups. In this section we discuss several possible heuristics for completeness checking, then evaluate their performance in the next section in a realistic setting based on Wikidata.

Before delving further into the discussion of heuristic approaches for completeness checking, we provide basic practical assumptions underlying the development of the heuristics. Our first assumption concerns the length of completeness statements. Similarly to queries, which in many practical cases consist of a limited number of triple patterns [15–17], we conjecture that also completeness statements would be of limited length. At the same time, it is conceivable that the number of completeness statements is large in practice, in particular since big KBs such as Wikidata may have many parts of data that are actually complete. Nevertheless, for a given query, most statements are likely to be irrelevant. For example, the statement "All players of Arsenal" is irrelevant to the query "Give founders

of the EU." Consequently, an efficient implementation should filter out such irrelevant statements. Moreover, as users are likely to provide completeness statements for similar topics, we introduce generic *completeness templates*. By providing a compact representation of completeness statements, completeness templates enable multiple statements to be processed simultaneously. We also provide a heuristic called *prioritized evaluation*, which tunes the way the crucial part is computed in data-aware completeness reasoning (cf., Eq. (1)).

In the next subsections, we first provide a heuristic technique for a simpler problem, that is, data-agnostic completeness checking, followed by several heuristic techniques for data-aware completeness checking.

6.1. Data-agnostic Completeness Checking

We now address a heuristic technique for data-agnostic completeness checking, originally proposed in [6], which can also be leveraged for pattern soundness checking, thanks to the characterization in Theorem 3. The theorem states that pattern soundness checking can be reduced to the (data-agnostic) checks of whether a set $\mathcal{C}$ of completeness statements entails conditional completeness statements of the form $Compl(P \mid P')$. By Proposition 10, such a check can be done by evaluating the union of the `CONSTRUCT` queries Q_C, for every completeness statement $C \in \mathcal{C}$, over the prototypical graph $\tilde{P} \cup \tilde{P}'$. A statement C contributes to this evaluation only if the result of Q_C over this graph is non-empty. Clearly, a necessary condition for C to contribute is that all terms in C (i.e., IRIs and literals) occur among the terms in $P \cup P'$, written $terms(C) \subseteq terms(P \cup P')$. We call such a C *term-relevant* for $Compl(P \mid P')$. We can retrieve all such term-relevant C by evaluating the subset query asking for the statement set $\{ C \in \mathcal{C} \mid terms(C) \subseteq terms(P \cup P') \}$.

A simple way (yet efficient, as we will show in Section 7) to implement such subset queries is to build a hashmap of all statements where the key of C is $terms(C)$. With this hashmap, we can answer the subset query by retrieving for all non-empty subsets $S \subseteq terms(P \cup P')$ the statements C with $terms(C) = S$. The completeness check can then be done only with these statements, which are potentially far fewer than the original statements.

Example 12. Consider the following set $\mathcal{C}_{org}$ of 10,000 completeness statements:

- $C_{lang} = Compl((?c, a, country), (?c, lang, ?l))$
- $C_{eu} = Compl((EU, founder, ?f))$
- $C_{org1} = Compl((org1, founder, ?f))$
- ...
- $C_{org9998} = Compl((org9998, founder, ?f))$.

The corresponding hashmap is as follows, where the keys are sets of terms occurring in the completeness statements, and the values are sets of completeness statements with those terms:

- $\{a, country, lang\} \mapsto \{C_{lang}\}$,
- $\{EU, founder\} \mapsto \{C_{eu}\}$,
- $\{org1, founder\} \mapsto \{C_{org1}\}$,
- ...
- $\{org9998, founder\} \mapsto \{C_{org9998}\}$.

Consider the query P_f from Section 3.2.2,

$$P_f = \{(?c, a, country), \neg\exists\{ (?c, lang, en) \}, \neg\exists\{ (EU, founder, ?c) \}\},$$

and let $P_1 = \{(?c, lang, en)\}$ (i.e., the BGP of the first `NOT-EXISTS` pattern) and $P_2 = \{(EU, founder, ?c)\}$ (i.e., the BGP of the second `NOT-EXISTS` pattern). To verify query soundness wrt. $\mathcal{C}_{org}$, according to Theorem 3 we check whether both $\mathcal{C}_{org} \models Compl(P_1 \mid P_f^+)$ and $\mathcal{C}_{org} \models Compl(P_2 \mid P_f^+)$. By applying $T_{\mathcal{C}_{org}}$ according to Proposition 10, we conclude that both entailments hold. However, rather than evaluating all 10,000 statements in $\mathcal{C}_{org}$ over the prototypical graph, we can now use the term hashmap to rule out irrelevant statements, as follows. Consider the first check. The set of terms from the BGP is $terms(P_1 \cup P_f^+) = \{ a, country, lang, en \}$. From these four terms, we generate 15 non-empty subsets: $\{a\}$, $\{country\}$, ..., and the set $terms(P_1 \cup P_f^+)$ itself. By looking up the statements for these subsets using the hashmap, we end up with the singleton set $\{ C_{lang} \}$. Note that the other 9,999 statements are irrelevant and thus are left out. By performing an analogous operation for the latter case, we retrieve the singleton $\{ C_{eu} \}$. Thus, instead of considering 10,000 statements in the reasoning, we now consider only 2.

A potential, theoretical drawback of this heuristic technique is that when queries are long, the number of term subsets generated for hashmap lookups would become large. More precisely, the number of term subsets grows exponentially with the query length (i.e., number of triple patterns). Nevertheless, as discussed above, we expect that queries in the real world are of limited length [15–17].

6.2. Data-aware Completeness Checking

For the data-aware setting, reasoning needs also access to the data graph. The previous heuristic 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 three heuristic techniques: completeness templates, partial matching, and prioritized evaluation. Their individual and combined impacts on reasoning time are reported in Section 7.

6.2.1 Completeness Templates Templates support users in creating completeness statements about similar topics, as they occur for instance in IMDb, which reports completeness for movie cast and crew,[20] or in OpenStreetMap, which uses a wiki to record the completeness of objects in different geographical areas.[21] A *completeness template* is a 3-tuple $\tau = (C, V_\tau, \Omega)$, where C is a completeness statement, $V_\tau \subseteq var(C)$ 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

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

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

A key part of the algorithm for data-aware completeness checking, given a statement set $\mathcal{C}$ 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 $\tilde{P}_0 \subseteq T_\mathcal{C}(\tilde{P} \cup G)$, where $T_\mathcal{C}$ is the transfer operator for $\mathcal{C}$. Given a set $\mathcal{T}$ of completeness templates, analogously to Eq. (1), such a part satisfies the equation

$$P_0 = P \cap \tilde{id}^{-1}(T_{\mathcal{C}_\mathcal{T}}(\tilde{P} \cup G)). \quad (2)$$

A baseline approach to compute P_0 in Eq. (2) is to instantiate templates to yield completeness statements, and then apply the $T_\mathcal{C}$-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 $T_\mathcal{C}$-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 $\tilde{P} \cup G$, and verify in a second step which of the resulting mappings are compatible[22] with the instantiations of the template τ. In this way, all instances of τ can be processed simultaneously. To formally represent the above idea, given a set $\mathcal{T}$ of completeness templates, a frozen BGP $\tilde{P}$, and a graph G, we define a template-based transfer operator

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

The above operator computes for each template τ the evaluation of the BGP P_τ over $\tilde{P} \cup G$, keeps only those mappings compatible with Ω, and then takes the union. The crucial point here is that instead of evaluating the whole set of (instantiated) completeness statements, we only have to evaluate the (potentially far fewer) completeness templates. By the definition of completeness templates and the construction of $T_\mathcal{T}$, the BGP P_0 in Eq. (2) can alternatively be computed using $T_\mathcal{T}$.

Proposition 14. *Given a BGP P, a graph G, and a set $\mathcal{T}$ of completeness templates, it is the case that*

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

The proposition holds basically because the mappings that result from the evaluation of a template over $\tilde{P} \cup G$ and are compatible with Ω are exactly the same as those of the instantiated completeness statements of the template that can be applied over $\tilde{P} \cup G$.

[20] See e.g., http://www.imdb.com/title/tt0105236/fullcredits
[21] See e.g., http://wiki.openstreetmap.org/wiki/Abingdon
[22] That is, when those mappings are projected to the meta-variables of the template, the projected mappings exist in Ω.

6.2.2 Partial Matching While the above heuristic technique concerns a data structure, called completeness templates, that bundles similar completeness statements, here we develop a heuristic for filtering out irrelevant completeness statements. In the previous data-agnostic heuristic technique, we perform so-called subset matching: for each statement, collect the set of terms of the BGP and check if this is a subset of the set of terms of the query BGP. This technique is no more applicable because a part of the BGP of a statement can also be matched to the data graph.

For data-aware completeness reasoning, it is sufficient for a completeness statement to be relevant for a query if the statement *partially* captures the BGP of the query, which in the worst case means to capture just a single triple pattern of the query. This is shown in the computation of the crucial part (Eq. (1)), where the transfer operator is evaluated over the union of the prototypical graph and the data graph, and then is intersected with the BGP of the query. Thus, the notion of relevance needs to be adapted for the data-aware case: A completeness statement may potentially be relevant for a query, if there is at least one triple pattern of the statement which is more general than a triple pattern of the query. The notion of generality is defined as follows: a triple pattern (s, p, o) is more general than another triple pattern (s', p', o') if in the corresponding position, say, the subject (and analogously for the predicate and object), either s is the same term as s' or s is a variable. Note that as opposed to the data-agnostic heuristic that supports the matching of the whole statement BGP, we now need to support matchings of single triple patterns. Such a matching of a single triple pattern of a BGP is called a *partial matching* of the BGP.

Let us first sketch a way how to retrieve such partially matched statements. Again, we rely on hashmaps. We use each triple pattern of a statement as a hashkey, by which the statement can be retrieved. Thus, a statement with three triple patterns, for example, can be retrieved in three different ways. To find statements 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.[23]

[23] Technically, the predicate can be a variable too. Yet, in relation to completeness statements, we believe that to be complete for all possible relationships between two entities might not be reasonable in practice. Hence, only non-predicate terms are generalized.

Let us formalize the above sketch. Our main goal here is partial matching: retrieving only completeness statements having a triple pattern that can potentially be mapped to a triple in a frozen BGP $\tilde{P}$. To this end, we first introduce the *signature operator* $\sigma(\cdot)$ that abstracts away concrete variables by replacing every occurrence of a variable with the reserved IRI `_var`. It can be applied to any syntactic object. As an illustration, the signature of the BGP $P_{usa} = \{ (usa, lang, ?l) \}$ is $\sigma(P_{usa}) = \{ (usa, lang, \texttt{\_var}) \}$.

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

$$M((s,p,o)) = \{ C \in \mathcal{C} \mid (s,p,o) \in \sigma(P_C) \}.$$

In practice, such a mapping can be realized by standard hashmaps. 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 $(usa, lang, \texttt{\_var})$ is the set $\{ (usa, lang, \texttt{\_var}), (\texttt{\_var}, lang, \texttt{\_var}) \}$.

Now, we are ready to define the operator `pmatch` that, given a set of statements $\mathcal{C}$, retrieves those elements of $\mathcal{C}$ that can potentially 'transfer' at least one triple in the frozen BGP $\tilde{P}$. Technically, `pmatch` maps P and $\mathcal{C}$ to the set of *partially matched* statements wrt. P and $\mathcal{C}$, denoted $\texttt{pmatch}(P, \mathcal{C})$, and 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, for each triple in the signature of the BGP P, the generalizations of that triple and then maps the generalizations to their corresponding statements in $\mathcal{C}$. By the construction of the mapping M and the generalization operator, it is the case that $\texttt{pmatch}(P, \mathcal{C})$ preserves the crucial-part operator in Eq. (1), as stated in Proposition 15.

Proposition 15. *Given a BGP P, a graph G, and a set $\mathcal{C}$ of completeness statements, it holds that*

$$\begin{aligned} cruc_{\mathcal{C},G}(P) &= P \cap \tilde{id}^{-1}(T_{\mathcal{C}}(\tilde{P} \cup G)) \\ &= P \cap \tilde{id}^{-1}(T_{\texttt{pmatch}(P,\mathcal{C})}(\tilde{P} \cup G)). \end{aligned}$$

This means that instead of taking all the statements in $\mathcal{C}$, it is enough to consider only the subset

pmatch$(P,\mathcal{C})$, which is potentially smaller than $\mathcal{C}$. The proposition holds since all applicable completeness statements over $\tilde{P} \cup G$ are actually partially matched statements (and by construction, all partially matched statements appear in $\mathcal{C}$).

Partial matching can also be leveraged immediately to support completeness templates: we simply take the (uninstantiated) statements of the completeness templates for building the hashmap in partial matching (that is, the C part in the template (C, V_τ, Ω)). The rest of the matching procedure can then be adapted accordingly.

6.2.3 Prioritized Evaluation An important operator in checking data-aware completeness is the crucial-part operator,

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

which computes the maximal part of a BGP containing variables that can be instantiated completely, which is needed for subsequent steps in the data-aware completeness checking procedure. In the above operator, CONSTRUCT queries of completeness statements are evaluated over the union of the prototypical graph and the data graph, and then (after the melting) are intersected with the query BGP. This means that to check whether the $T_{\mathcal{C}}$-operator maps a triple in $\tilde{P}$ by a statement C, we first evaluate P_C over the union graph $\tilde{P} \cup G$, with the condition that at least one triple pattern in P_C is mapped to a triple in $\tilde{P}$ (since otherwise the mapping does not have any contribution). Here, instead of directly evaluating the CONSTRUCT query Q_C over the union graph, which can be large in size (due to G), but may potentially waste computation time, what if the evaluation is done by prioritizing the part $\tilde{P}$ and then, only if Q_C can be partially mapped to some parts of $\tilde{P}$, the remaining parts of Q_C are evaluated over the data graph. This way, not all Q_C's are evaluated over the data graph, but only those that may be useful for guaranteeing the completeness of parts of the BGP.

To formalize the above idea, we 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* graph and the second as the *optional* graph, which means that at least one triple pattern of the BGP is mapped to a triple in 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 $[\![P]\!]_{(G_1,G_2)} = \{\mu \in [\![P]\!]_{G_1 \cup G_2} \mid \mu P' \subseteq G_1 \text{ for some } P' \subseteq P, P' \neq \emptyset\}$. In the case of completeness checking, the mandatory graph will be the frozen BGP $\tilde{P}$ and the optional graph will be the data graph G. In our transfer operator, prioritized evaluation is performed (instead of the standard evaluation) when the CONSTRUCT queries of the statements are evaluated over the prototypical graph and the data graph. Note that by the definition of prioritized evaluation and the crucial operator, it is the case that both evaluation methods (i.e., the prioritized evaluation and the prioritized evaluation) produce the same result.

Example 13. Consider the BGP $P_{usa} = \{(usa, lang, ?l)\}$, asking for languages of the USA, the graph

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

and the completeness statement

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

Then, with P_C as the BGP of the statement C, we have that $[\![P_C]\!]_{(\tilde{P}_{usa}, G_{org})} = \{\{?c \mapsto usa, ?lang \mapsto \tilde{l}, ?org \mapsto org_2\}\}$. The reason is that in a prioritized evaluation, at least one triple of P_C has to be matched to the triple in $\tilde{P}_{usa}$. The only way to do that is to map $?c \mapsto usa$ and $?lang \mapsto \tilde{l}$. This leaves only one possibility to match $(?org, founder, ?c)$ to G_{org}.

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

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

set. Formally, we compute the union

$$\bigcup_{\substack{P'_C \subseteq P_C \\ P'_C \neq \emptyset}} \bigcup_{\substack{\lambda \in [\![P'_C]\!]_{\tilde{P}} \\ \lambda \text{ is pure}}} \{\lambda\} \bowtie [\![\lambda(P_C \setminus P'_C)]\!]_G,$$

which equals $[\![P_C]\!]_{(\tilde{P},G)}$. The equality holds for the reason discussed: the union above considers all possible subsets of P_C (except the empty set), and non-pure instantiations cannot be evaluated over the data graph G.

The above discussion concerns the prioritized evaluation of the transfer operator that uses plain completeness statements. For completeness templates, the template-based transfer operator $T_{\mathcal{T}}$ as in Section 6.2.1 is slightly modified to account for a prioritized evaluation in the way the BGP of the template's statement is evaluated, as follows, where $\mathcal{T}$ is a set of completeness templates, $\tilde{P}$ is a frozen BGP and G is a graph:

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

As shown above, instead of evaluating the BGP P_τ of a template $\tau \in \mathcal{T}$ using the standard evaluation over the union of $\tilde{P}$ and G, the prioritized evaluation is used instead.

Trade-offs of the Heuristics A potential benefit of the templates are that they can be used to group (possibly many) similar completeness statements into one, and hence we can reduce the number of evaluations in the transfer operator. A potential drawback of using templates are that since their BGPs are by construction more general than completeness statements, the evaluation of the template's BGP over the data graph may return many results which have to be checked for compatibility wrt. the set Ω of mappings of the template.

As for partial matching, in principle it may filter out many completeness statements (and templates). Still, a potential drawback is that when the statements contain many overlapped parts to each other, then partial matching may consider many partially matched completeness statements, too. Hence, the reduction of the number of completeness statements that need to be considered in completeness reasoning might not be substantial.

As for prioritized evaluation, using it alone can still be expensive when there are a large number of completeness statements that need to be considered. Combining prioritized evaluation with completeness templates may provide a remedy to the generality of the BGPs of the completeness templates, since in this case, the BGPs of the templates have to be checked first if they may capture parts of the query, and only if so, they can be evaluated over the graph.

In Section 7, we will study how such trade-offs may behave in a practical setting, for all possible combinations of heuristics (that is, they may be applied alone, or together).

7. Experimental Evaluation

In this section, we report on our experimental evaluation of query completeness and query soundness reasoning. We have proposed in Section 6 three heuristic techniques for completeness reasoning, namely completeness templates, partial matching, and prioritized evaluation. Consequently, the practical impact of these techniques (and their interplay) on completeness reasoning needs to be validated. Furthermore, in the light of the query soundness problem, we want to study the performance of answer and pattern soundness checking in a realistic setting. The goal of our experimental evaluation is therefore two-fold:

1. To study how effective are completeness templates, partial matching, and prioritized evaluation, as well as their combinations, for completeness reasoning; and
2. to show the feasibility of soundness reasoning (which in turn reduces to completeness reasoning) in a realistic setting.

7.1. Query Completeness Evaluation

In this subsection, we first describe the experimental setup, followed by the discussion of the results of the experiments.

7.1.1. Experimental Setup

Given the three heuristic techniques for completeness reasoning, that is, completeness templates (`temp`), partial matching (`match`), and prioritized evaluation (`peval`), we want to analyze to which degree those techniques may provide speed-up to completeness reasoning. Specifically, not only do we want to study the effects of every individual heuristic, but also those of the combinations between different heuristics. We therefore distinguish between eight different heuristic settings: `no`, `temp`, `match`, `peval`, `temp+match`, `temp+peval`, `match+peval`, and `all`. The `no` is the

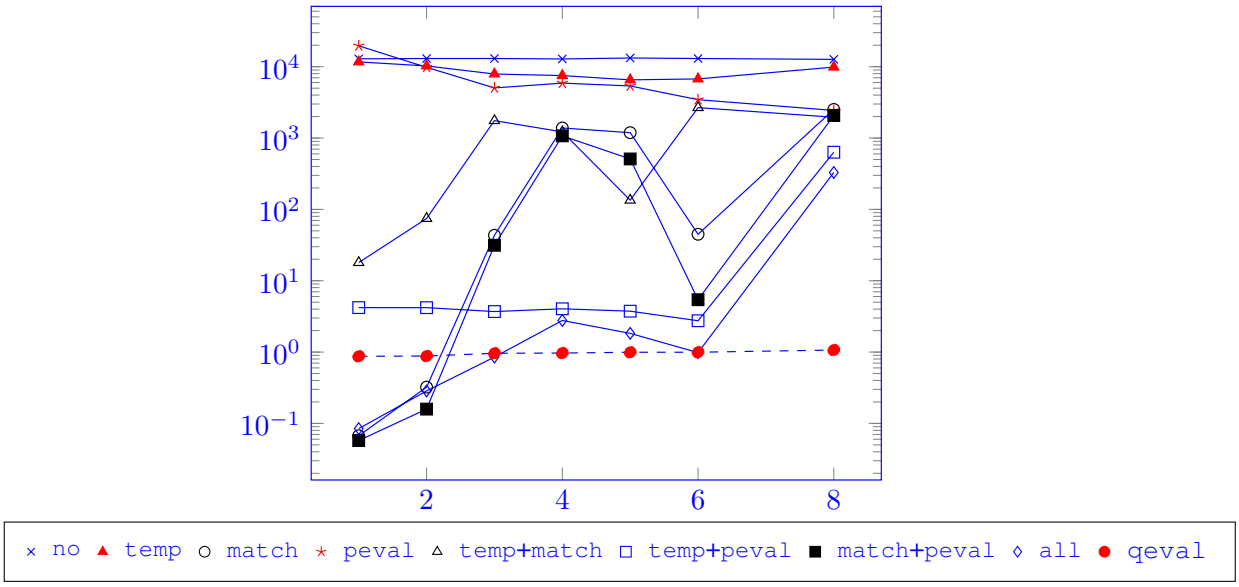

Fig. 4. Comparison of completeness checking using different heuristic settings. The x-axis is the query length, and the y-axis is the runtime in ms.

baseline setting with no heuristic (that is, the vanilla implementation of completeness reasoning), whereas the `all` is the setting with all heuristics.

As for completeness reasoning, there are three ingredients: graph, queries, and completeness statements. While machine-readable completeness statements in the wild are yet to appear, we aim to make our setup as realistic as possible. We take Wikidata as the graph for our evaluation. More specifically, we use the monolingual, direct-statement fragment (i.e., the fragment without qualifiers, references and non-English labels) of the Wikidata dump released in April 2018. The graph consists of around 491 mio triples (= 57 GB in the uncompressed NT format). We choose Wikidata mainly because of its popularity and good quality.

As for the experiment queries, they have to use the same vocabulary as the Wikidata graph described above. We therefore generate the experiment queries based on human-made, openly available queries on the Wikidata query page.[24] Since our experiment queries have to be conjunctive in order to be able to be checked for completeness, we extract the BGPs of the queries in the Wikidata query page and generate our experiment queries based on these BGPs. We cannot directly use these BGPs as our experiment queries due to the limited number of the extracted BGPs. Nevertheless, these BGPs may serve as a starting point to generate our experiment queries, in the following way: *(i)* for each extracted BGP, we evaluate it over the Wikidata graph; *(ii)* we take randomly 20 of the mappings, projected on the first variable of the BGP;[25] and *(iii)* we generate experiment queries by instantiating the BGPs with these projected mappings. We generate in total 1,211 queries with the average query length (i.e., the number of triple patterns) of 2.7. Note that this average query length reflects that of real-world queries [17].

Now, for the completeness statements, we generate them in a similar way as we generate the experiment queries. The only difference is that, in the second step above, instead of taking randomly 20 of the mappings, we obtain randomly 50% of the mappings to create completeness statements. This is to ensure the generation of a large number of completeness statements. In this setting, we also naturally represent completeness statements by completeness templates as follows: we take the extracted BGP as the template's BGP, and the projected mappings as the template's mappings. We generate in total 693,928 completeness statements with the average statement length (i.e., the number of triple patterns in the BGP of completeness statements) of 2.4. Moreover, these statements are represented by 65 completeness templates.

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

[25] If the number of mappings is below 20, then we just take all the mappings.

Implementation The reasoning program and experiment framework are implemented in Java using the Apache Jena library.[26] The graph of the experimental evaluation is loaded into a Jena TDB triple store. We measure the runtime of completeness reasoning with different heuristic settings, as described above, and of query evaluation. For each query length, we randomly sample 20 queries to be observed for the runtime, and take the runtime median between these 20 queries. The experiments are done on a PC with an Intel Core i7 3.6 GHz-processor, a 16 GB memory, and an HDD of 1 TB.

7.1.2. Results and Discussion

Figure 4 summarizes the results of the experiments. The figure shows the runtime comparison of completeness checking using different heuristic settings. The x-axis is the query length, whereas the y-axis shows the runtime in ms, and is in log-scale. We also add the query evaluation time to provide an orientation as to how completeness checking may compare to query evaluation.

In general, we observe that completeness checking using all heuristics (that is, the `all` setting) beats all the other heuristic settings. The `no`, `temp`, and `peval` settings are the worst-performing. Using the `temp` and `peval` heuristics alone does not help in speeding up completeness reasoning, as they only add a slight improvement. In the `temp` setting, despite the lower number of templates in comparison to the number of statements, completeness templates are by design less selective than completeness statements, and thus tend to be longer to compute for the transfer operator. The `peval` heuristic does not perform well as anyway all completeness statements have to be evaluated. Interestingly, the `match` gives a mixed impression: for short queries, `match` seems to be a sufficient heuristic for completeness checking; yet, when the queries become longer, starting from those of length 3, its runtime increases considerably. The only exception in the queries of length 6 might be due to the following reason: in our experiment setup there are only 5 queries that are of length 6, generated from one extracted BGP only. Hence, those queries might be queries that do not have many overlaps with the other queries, which is an advantage to the `match` technique (recall that our completeness statements are generated in a similar way as the queries). The queries of length 8 (which are not many in our experiment query set) are in fact the opposite of the queries of length 6: that they have many overlaps with the other queries, hence the runtime increase for queries of length 8.

Let us now observe the performance of combining the heuristics. Adding partial matching on top of completeness templates provides a runtime improvement, as can be seen from the `temp+match`. Here, only partially matched templates are evaluated instead of all templates. For the case of `temp+peval`, we observe that completeness templates are reasonable for completeness checking when used together with prioritized evaluation. This is because prioritized evaluation tends to avoid the heavy computation of transfer operator over the graph whenever: (*i*) no parts of the template can be applied to the prototypical graph; or (*ii*) parts of the template can be applied to the prototypical graph, but capturing frozen variables (which do not exist in the graph). Still, all templates have to be considered, since there is no partial matching. The addition of prioritized evaluation to partial matching only provides a slight improvement, as observed in the `match+peval`. Combining all the heuristic techniques, as in the `all` setting, gives the best performance. The reason is that in addition to the benefits of the `temp+peval`, as described above, we also obtain the effect of partial matching in filtering out irrelevant completeness templates, hence only potentially relevant templates are considered.

Overall, our experimental evaluation has answered the question as to how effective are heuristic techniques (and their combinations) for completeness checking. Partial matching, though seeming promising for short queries, tends to have a negative effect for long queries. Templates and prioritized evaluation are best applied together. Using all the three heuristics brings the advantages of each heuristic into one.

7.2. Query Soundness Evaluation

Here, we describe the setup of the experiments, and then discuss the results of the experiments.

7.2.1. Experimental Setup

From the characterizations in Section 5, we are able to check query soundness by reducing it to query completeness checking. Pattern soundness checking can be reduced to data-agnostic completeness checking, for which we proposed a heuristic technique in Section 6.1 that is based on the term-relevance principle. As for answer soundness checking, we also reduce it to the problem of data-aware completeness check-

[26] http://jena.apache.org/

ing. In Section 7.1, we have shown that not only using all the heuristics of completeness templates, partial matching, and prioritized evaluation yields runtime improvements over a plain implementation of a completeness reasoner, but also that the approach where all these heuristics are applied in combination performs the best. Given these improvement techniques for completeness checking, we want to study the feasibility of soundness checking in a realistic setting. Additionally, we want to know how pattern soundness checking compares to answer soundness checking, and how soundness checking compares to query evaluation.

For experimentally evaluating query soundness, we (again) require three ingredients: graph, queries, and completeness statements. We use the Wikidata graph that is described as before in the experiments in Section 7.1. For the experiment queries, now we need queries with negation. Similarly as before, we take Wikidata queries,[27] and extract their BGPs to generate experiment queries, this time with negation. We want to have queries with negation of various shapes. For this reason, from the BGPs of the Wikidata queries we generate 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 is set to at most three, since most real-world queries are of length up to three [15]. We project out all variables in the positive part to correspond to graph pattern evaluation (recall the definition of graph pattern in Section 2.1). The statistics of the generated queries are shown in Table 2. The number of queries across different cases ranges from 18 to 59. The median query length is 3 for all the cases, except the $\mathbf{Q}_{\texttt{threeTPs}}$ case with query length of 4.

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 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.

In this setting, we can naturally represent completeness statements by completeness templates (see Section 6.2). 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 generated completeness statements in an additional way, which differs from 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[28] 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. The statistics of the generated completeness statements are shown in Figure 2. The number of statements ranges from around 61,000 to 454,000.

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

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

[28] We fixed an ordering.

– `threeTPsTO` ('TO' for together) is where the last three triple patterns are negated together and the statements are for the whole BGP.

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 15 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 490,000 results). The experiments were done on a PC with an Intel Core i7 3.6 GHz-processor, a 16 GB memory, and an HDD of 1 TB.

7.2.2. Experimental Results and Discussion

Table 2 summarizes the results of the soundness reasoning experiments for five cases. In the table, query evaluation time ranges from 20 ms to 89 ms. Pattern soundness checking always takes less than 0.3 ms. The term-relevance principle is likely to help rule out irrelevant completeness statements before performing the actual pattern soundness checking. Note that the pattern soundness checking time for the `oneTPoneTP` case takes the longest due to two data-agnostic completeness checks that have to be performed for every pattern soundness checking (recall the reduction in Theorem 3). In comparison to query evaluation, pattern soundness checking is much faster, particularly due to its data-agnostic behavior.

As for answer soundness checking, we observe that the runtime ranges from 2 ms to 592 ms. The `threeTPsTO` takes longer due to the large number of query answers that have to be checked for soundness. When we break down the time per answer, the computation takes less than 0.4 ms. Also, it is likely that the more triple patterns there are in the negation part, the longer the soundness check per answer takes. In comparison to query evaluation, answer soundness checking is quite competitive. Yet, when compared to pattern soundness checking, answer soundness checking is slower due to its data-aware behavior.

Let us look more closely at answer soundness checking. Figure 5 shows the comparison between the number of query answers, query evaluation time, and answer soundness checking time for cases `oneTP`, `twoTPsTO`, and `threeTPsTO`. We omitted the figure of `oneTPoneTP` since it is similar to that of `oneTP`, and of `twoTPsSE` as it is similar to that of `twoTPsTO`. 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 ms 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 all the cases: At first, when query answers are just a few, 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. When queries become more complicated, the cross-over point happens earlier. 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 evaluation for soundness checking over a realistic setting based on Wikidata. Soundness checking using the heuristic techniques as proposed in Section 6 has been validated to be feasible for our experiment cases. To get an idea of how long soundness checking may take without using any heuristics, we perform vanilla answer soundness checking for the `oneTP` case, and it can take more than 300 s for just a single query. In comparison to query evaluation and answer soundness checking, pattern soundness checking runs much faster. We would recommend that in practice, before applying answer soundness checking, pattern soundness checking should be done first since it takes less time, and by Proposition 2, if pattern soundness holds, then all answers are sound.

8. Related Work

In this section, we discuss related work for query completeness and query soundness.

Table 2

The number of statements $|\mathcal{C}|$, the number of queries N_Q, 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 ms.

Case	$\lvert\mathcal{C}\rvert$	N_Q	$\lvert Q\rvert$	$\lvert[\![Q]\!]_G\rvert$	t_Q	t_{AS}	$t_{AS/a}$	t_{PS}
`oneTP`	61,971	59	3	51	21	2.03	0.043	0.13
`oneTPoneTP`	454,131	30	3	374	89	17.4	0.046	0.21
`twoTPsTO`	395,399	41	3	198	20	26.5	0.15	0.13
`twoTPsSE`	414,321	41	3	198	20	12.7	0.06	0.13
`threeTPsTO`	335,852	18	4	2,671	89	592	0.39	0.15

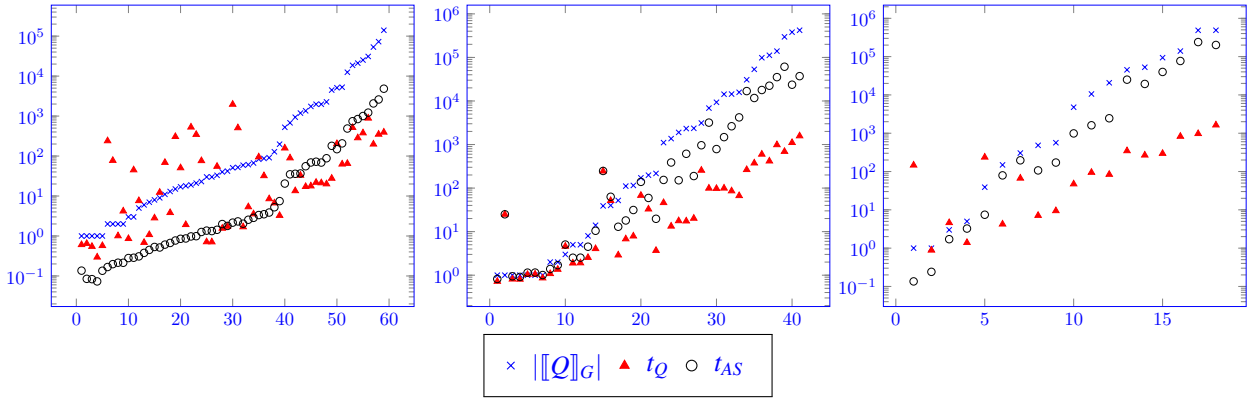

Fig. 5. Comparison between the number of query answers ($|[\![Q]\!]_G|$), query evaluation time (t_Q), and answer soundness checking time (t_{AS}) for the experiment cases: `oneTP`, `twoTPsTO`, and `threeTPsTO`. The x-axis is for query rank (from the lowest to the highest of number of query answers) and the y-axis is for both number of answers and runtime in ms.

Query Completeness Data completeness concerns the breadth, depth, and scope of information [18] and is deemed to be one of the most significant data quality dimensions [19]. In the field of relational databases, Motro [20] and Levy [21] were among the first to investigate data completeness. Motro modeled completeness of an available database instance in terms of its relationship with a hypothetical, unknown instance that represents the real world. This model has been taken up in the present paper by the notion of extension pairs. Motro also defined completeness constraints, which correspond to our completeness statements and developed a sound technique to check query completeness of conjunctive queries in the data-agnostic setting. Levy generalized completeness constraints to local completeness statements, which correspond to the conditional statements in Section 3.2.2 and are more expressive than the statements in the framework studied here.

Razniewski and Nutt [22] further extended this work by reducing completeness reasoning to containment checking, for which many algorithms are known, and by characterizing the complexity of reasoning for different classes of conjunctive queries. While their investigations were largely at the schema level (that is, data-agnostic), they also showed that for conditional statements the combined complexity of completeness checking in the data-aware setting is Π_2^P-complete and the data complexity is polynomial. Since conjunctive SPARQL queries can be seen as a special case of conjunctive queries in the relational framework, and since our completeness statements are unconditional, the Π_2^P-hardness result in the present paper strengthens the one in [22], while the result on data complexity can be deduced from theirs.

This line of work on data-agnostic reasoning was later continued by Darari et al. [4, 6], incorporating orthogonal aspects such as time-awareness and federation. In contrast, the present work focuses on more restrictive, yet also more intuitive unconditional completeness statements, for which it provides an in-depth complexity analysis and implementation techniques for data-aware reasoning. The problems of answer and pattern soundness are also completely new. In [23],

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 checking. In our work, a sound and complete algorithm for data-aware completeness checking and a comprehensive complexity analysis of the checking are given.

On a more abstract level, Fürber and Hepp [24] 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 [25]. 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. [26] 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. [27] 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). In the context of crowdsourcing, Chu et al. [28] 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; whereas Acosta et al. [29] developed HARE, a hybrid SPARQL engine to enhance answer completeness. 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 concentrate more on increasing the degree of KB and query completeness.

Galárraga et al. [30] 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. [31] (called Local Closed-World Assumption in their paper) to develop Knowledge Vault, a Web-scale system for probabilistic knowledge fusion. Our completeness statements are in fact a generalization of the assumption used in the above work.

Query Soundness The use of negation in querying can be traced back to Codd's relational calculus [32], where a tuple is included in the complement of a relation if it is not explicitly given in the relation. Reiter [33] and Clark [34] generalized this 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 [5, 35]. However, since the semantics of RDF imposes the open-world assumption (OWA) [2], there remains a conceptual mismatch when negation in SPARQL over RDF datasets is evaluated in a closed-world style. In other words, there is a missing gap between the normative semantics of negation in SPARQL, which is based on the negation-as-failure ('negation from the failure to find a proof of the fact') [36], and the classical negation ('the negated fact truly holds') [37] due to RDF's openness. Moreover, 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. [38] 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. In description logics (DLs), Lutz et al. [39] proposed closed predicates, that is, concepts and roles that are interpreted to be complete, to enable a combination between open- and closed-world reasoning. A similar concept was also employed by Analyti et al. [40]. They 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 these two approaches, 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 in the sense that we can use BGPs to capture completeness. From the practical side of the Semantic Web, negation is featured in test queries of many popular SPARQL benchmarks such as SP^{2}Bench [41], Berlin SPARQL Benchmark (BSBM) [42], and FedBench [43], in which the closed-world assumption (CWA) is employed. Our work does not only provide formalizations, but also optimization techniques for checking the soundness of queries with negation, for which we have experimentally shown to improve the feasibility of the soundness checking in realistic settings based on Wikidata.

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.

Weak Monotonicity The open-world semantics of RDF has led to the question which type of queries appropriately reflects this fact. Arenas and Pérez noted that for positive graph patterns, that is, patterns composed by the operators AND, UNION, and FILTER, the semantics of SPARQL reflects the open-world semantics in that the answers for such a pattern are exactly the certain answers [45].

As an additional feature that supports querying incomplete information, SPARQL offers the OPT constructor that binds variables if they can be matched to the data, and leaves them unbound otherwise. Patterns with OPT are not necessarily monotone because adding triples to a data graph may lead to bindings for variables that were not bound previously. Such patterns may however be *weakly monotone*, that is, for every answer mapping returned over the smaller graph there is an answer over the larger graph that extends it. Thus, weakly monotone patterns do not not lose information when new information becomes available, although they may not preserve the shape of answers.

While not all patterns with OPT are weakly monotone, this is the case for the class of well-designed patterns [45]. It has been shown, though, that there also weakly monotone patterns beyond the well-designed patterns. Arenas and Ugarte have investigated patterns that are weakly monotone over potentially infinite graphs and characterized them by applying interpolation techniques from first-order logic [46].

In the presence of incomplete data, OPT can be interpreted in two different ways. For example, the pattern $\{ (?p, \mathit{livesIn}, \mathit{bolzano}) \text{ OPT } (?p, \mathit{hasEmail}, ?e) \}$ can be interpreted as asking for all persons living in Bolzano, together with the email addresses present in the data, or for all such persons together with all their email addresses, if they have any. Over complete data, where no information is added, only the second interpretation is meaningful.

Darari et al. have characterized completeness of well-designed patterns in the data-agnostic setting [4, 6]. We conjecture that techniques from that work and from the current paper can be combined to reason about the completeness of well-designed patterns in the presence of data.

9. Discussion

Here we discuss issues related to our framework: creation of completeness information, no-value information, blank nodes, and RDFS extension.

Creation 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 example on Wikipedia, IMDb, and OpenStreetMap. 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 were collected in [47]. Galárraga et al. [48] investigated various signals, such as popularity, update frequency, and cardinality, that can be used to identify complete parts of a KB via rule-mining techniques. Mirza et al. [49, 50] developed techniques for relation cardinality extraction from text, which can be leveraged to generate completeness statements in the following way: when the extracted car-

dinality of a relation matches with the relation count in a KB, then a completeness statement can be generated. COOL-WD is a collaborative, web-based system for managing and consuming completeness information about Wikidata, which currently stores over 10,000 real completeness statements [51], and is available at http://cool-wd.inf.unibz.it. Additionally, COOL-WD demonstrates how provenance information, such as authorship, timestamp, and external reference, can be added to completeness statements, which can then serve as a basis for trust determination over query completeness and soundness 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").

No-Value Information Completeness statements can also be used to represent no-value information. Such information is particularly useful to distinguish between a value that does not exist due to data incompleteness or due to its inapplicability. Wikidata, for instance, contains about 19,000 pieces of no-value information over 269 properties.[29] In our motivating scenario, there is the completeness statement about the official languages of the USA with no corresponding data in the graph. In this case, we basically say that the USA has no official languages. As a consequence of having no-value information, we can be complete for queries despite having the empty answer. Such a feature is similar to that proposed in [52]. The only difference is that here we need to pair completeness statements with a graph that has no corresponding data captured by the statements, while in that work, no-value statements are used to directly say that some parts of data do not exist.

SPARQL Fragment In this paper we have focused on conjunctive SPARQL, that is, SPARQL without `UNION`, `OPTIONAL` , or arithmetic comparisons. Conjunctive SPARQL is the foundational fragment underlying all extensions [9, 10], thus we believe that the study of completeness and soundness reasoning for this fragment provides a solid theoretical basis for further analysis of richer fragments. Our main results can be translated to richer fragments as sufficient conditions for completeness and soundness. We conjecture that also necessary conditions can be established in analogy to the data-agnostic case [6], for `UNION` in terms of completeness and soundness of the disjuncts, or for well-designed patterns with `OPTIONAL` in terms of completeness and soundness of pattern trees [53], and for patterns with arithmetic comparisons by the use of representative orderings for the containment checks [54].

Blank Nodes The use of blank nodes in RDF has been a controversial topic in the Semantic Web community [55, 56]. In Linked Data applications, blank nodes add complexity to data processing and data interlinking due to the local scope of their labels [57, 58]. 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 [59]. Nevertheless, blank nodes are used in practice to some degree: *(i)* for modeling unknown nulls [52, 59], and *(ii)* for modeling n-ary relations as auxiliary instances in reification [60].

For the former usage, completeness of a topic that contains a blank node is contradictory, as we argue that completeness statements should capture only "known and complete" information. For instance, 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 [2, 61, 62], 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 as per [59]). Nevertheless, in practice Semantic Web developers often tend to directly use IRIs instead of blank nodes for representing auxiliary resources, for instance by Wikidata [58].[30]

RDFS Extension RDFS [63] adds lightweight semantics to describe the structure and interlinking of data, usually sufficient for Linked Data publishers [57]. Main RDFS inference capabilities consist of class and property hierarchies, as well as property do-

[29] as of Feb 18, 2017

[30] 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.

mains and ranges [57, 64], which are widely used in practice [65]. Darari et al. [4] formalized the incorporation of RDFS in data-agnostic completeness reasoning. Using a similar technique as in [4], 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 Section 4.1) via the closure operation wrt. RDFS ontologies [64]. More precisely, in the crucial part, the closure has to be computed before and after the T_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 10). For answer soundness checking, we can simply rely on the data-aware completeness checking with RDFS incorporation we just sketched. In summary, the addition 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 [64].

Extending the Completeness and Soundness Characterizations to Relational Databases Our completeness and soundness results are based on RDF and SPARQL, which are the triple-based data model and triple-pattern-based query language for the Semantic Web. However, analogous algorithms, characterizations and complexity results can also be formulated and proved for relational databases if we generalize triples to n-ary relations and conjunctive patterns to projection-free conjunctive queries. The completeness statements in this paper can generalized in two ways. One way is to view them as so-called query completeness statements, which state that a given query is complete (see [66]), where the query in this case is projection-free. Another way is to see them as collections of local completeness statements (see [21, 22]). The proofs for the analogous results would be similar to the ones given here. This class of completeness statements, however, has not yet considered in the relational setting, while they arise naturally in the Semantic Web context.

10. Conclusions and Future Work

The open-world assumption of RDF and the closed-world evaluation of SPARQL have created a gap on how we should treat the completeness and soundness of query answers. This paper bridges the gap between RDF and SPARQL via completeness statements, metadata about (partial) completeness of RDF data sources. In particular, we have introduced the problem of query completeness checking over RDF data annotated with completeness statements. We also developed an algorithm and performed a complexity analysis for the completeness problem. Then, we formulated the problem of soundness for SPARQL queries with negation, and characterized it via a reduction to completeness checking. We proposed optimizations for completeness checking, and provided experimental evidence that our techniques are feasible in realistic settings over Wikidata for both query completeness and soundness problems.

Our current approach tackles the core fragment of SPARQL, and can be easily adapted to provide sufficient characterizations of richer fragments (e.g., involving union, arithmetic filter, or—for queries with negation—the selection operator), setting a solid basis for future investigations into the full characterization of those fragments. 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 students who were born after 1991 and who do not speak German." 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. Another 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 government. The purpose is to analyze whether our approach is sufficient or not for their requirements, and if not, on which side it can be improved. Last but not least, we want to study how the soundness of queries with the `MINUS` negation can also be characterized. We conjecture that our results apply also to queries with the `MINUS` negation where there is a shared variable between the positive part and each of the negative parts.

References

[1] F. Darari, S. Razniewski, R.E. Prasojo and W. Nutt, Enabling Fine-Grained RDF Data Completeness Assessment, in: *Web Engineering - 16th International Conference, ICWE 2016, Lugano, Switzerland, June 6-9, 2016. Proceedings*, 2016, pp. 170–187.

[2] P.J. Hayes and P.F. Patel-Schneider (eds), *RDF 1.1 Semantics*, W3C Recommendation, 25 February 2014.

[3] D. Vrandecic and M. Krötzsch, Wikidata: a free collaborative knowledgebase, *Commun. ACM* **57**(10) (2014), 78–85.

[4] F. Darari, W. Nutt, G. Pirrò and S. Razniewski, Completeness Statements about RDF Data Sources and Their Use for Query Answering, in: *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, 2013, pp. 66–83.

[5] S. Harris and A. Seaborne (eds), *SPARQL 1.1 Query Language*, W3C Recommendation, 21 March 2013.

[6] F. Darari, W. Nutt, G. Pirrò and S. Razniewski, Completeness Management for RDF Data Sources, *ACM Trans. Web* **12**(3) (2018), 1–53.

[7] semantic-web@w3.org Mail Archives: Open world issue (opening vs closing days) and SPARQL CONSTRUCT, Accessed: 2017-01-15 from https://lists.w3.org/Archives/Public/semantic-web/2008May/0152.html.

[8] semantic-web@w3.org Mail Archives: The Open world assumption shoe does not always fit - was: RE: [ontolog-forum] Fwd: Ontolog invited speaker session - Dr. Mark Greaves on the Halo Project - Thu 2008.06.19, Accessed: 2017-01-15 from https://lists.w3.org/Archives/Public/public-semweb-lifesci/2008Jun/0084.html.

[9] E. Liarou, S. Idreos and M. Koubarakis, Evaluating Conjunctive Triple Pattern Queries over Large Structured Overlay Networks, in: *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings*, 2006, pp. 399–413.

[10] G. Stefanoni, B. Motik and E.V. Kostylev, Estimating the Cardinality of Conjunctive Queries over RDF Data Using Graph Summarisation, in: *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, 2018, pp. 1043–1052.

[11] F. Darari, S. Razniewski and W. 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.*, 2014, pp. 269–272.

[12] J. Pérez, M. Arenas and C. Gutierrez, Semantics and complexity of SPARQL, *ACM Trans. Database Syst.* **34**(3) (2009).

[13] M.Y. Vardi, The Complexity of Relational Query Languages (Extended Abstract), in: *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, 1982, pp. 137–146.

[14] A.K. Chandra and P.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.

[15] M. Arias, J.D. Fernández, M.A. Martínez-Prieto and P. de la Fuente, An Empirical Study of Real-World SPARQL Queries, in: *Proceedings of the 1st International Workshop on Usage Analysis and the Web of Data (USEWOD'11)*, 2011.

[16] L. Rietveld and R. Hoekstra, Man vs. machine: Differences in SPARQL queries, in: *Proceedings of the 4th USEWOD Workshop on Usage Analysis and the Web of Data, ESWC, Crete, Greece*, 2014.

[17] M. Saleem, M.I. Ali, A. Hogan, Q. Mehmood and A.N. Ngomo, LSQ: The Linked SPARQL Queries Dataset, in: *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II*, 2015, pp. 261–269.

[18] R.Y. Wang and D.M. Strong, Beyond Accuracy: What Data Quality Means to Data Consumers, *J. of Management Information Systems* **12**(4) (1996), 5–33.

[19] C. Batini and M. Scannapieco, *Data and Information Quality - Dimensions, Principles and Techniques*, Data-Centric Systems and Applications, Springer, 2016.

[20] A. Motro, Integrity = Validity + Completeness, *ACM Trans. Database Syst.* **14**(4) (1989), 480–502.

[21] A.Y. Levy, Obtaining Complete Answers from Incomplete Databases, in: *VLDB*, Morgan Kaufmann, 1996, pp. 402–412.

[22] S. Razniewski and W. Nutt, Completeness of Queries over Incomplete Databases, *PVLDB* **4**(11) (2011), 749–760.

[23] S. Razniewski, F. Korn, W. Nutt and D. Srivastava, Identifying the Extent of Completeness of Query Answers over Partially Complete Databases, in: *ACM SIGMOD 2015*, 2015, pp. 561–576.

[24] C. Fürber and M. Hepp, SWIQA - a Semantic Web Information Quality Assessment Framework, in: *Proceedings of the 2011 European Conference on Information Systems (ECIS), Helsinki, Finland, June 9-11, 2011*, 2011.

[25] A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann and S. Auer, Quality assessment for Linked Data: A Survey, *Semantic Web* **7**(1) (2016), 63–93.

[26] P.N. Mendes, H. Mühleisen and C. Bizer, Sieve: linked data quality assessment and fusion, in: *Proceedings of the 2012 Joint EDBT/ICDT Workshops, Berlin, Germany, March 30, 2012*, 2012, pp. 116–123.

[27] I. Ermilov, J. Lehmann, M. Martin and S. 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*, 2016, pp. 38–46.

[28] X. Chu, J. Morcos, I.F. Ilyas, M. Ouzzani, P. Papotti, N. Tang and Y. Ye, KATARA: A Data Cleaning System Powered by Knowledge Bases and Crowdsourcing, in: *ACM SIGMOD 2015*, 2015, pp. 1247–1261.

[29] M. Acosta, E. Simperl, F. Flöck and M. Vidal, HARE: A Hybrid SPARQL Engine to Enhance Query Answers via Crowdsourcing, in: *K-CAP 2015*, 2015, pp. 11–1118.

[30] L.A. Galárraga, C. Teflioudi, K. Hose and F.M. Suchanek, AMIE: Association Rule Mining under Incomplete Evidence in Ontological Knowledge Bases, in: *WWW 2013*, 2013, pp. 413–422.

[31] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun and W. Zhang, Knowledge vault: a web-scale approach to probabilistic knowledge fusion, in: *ACM SIGKDD 2014*, 2014, pp. 601–610.

[32] E.F. Codd, Relational Completeness of Data Base Sublanguages, *In: R. Rustin (ed.): Database Systems* (1972).

[33] R. Reiter, On Closed World Data Bases, in: *Logic and Data Bases*, H. Gallaire and J. Minker, eds, Springer US, Boston, MA, 1978, pp. 55–76.

[34] K.L. Clark, Negation as Failure, in: *Logic and Data Bases*, 1978, pp. 113–141.

[35] E. Prud'hommeaux and A. Seaborne (eds), *SPARQL Query Language for RDF*, W3C Recommendation, 15 January 2008.

[36] R. Reiter, Towards a Logical Reconstruction of Relational Database Theory, in: *On Conceptual Modelling (Intervale)*, 1982, pp. 191–233.

[37] M. Gelfond and V. Lifschitz, Classical Negation in Logic Programs and Disjunctive Databases, *New Generation Comput.* **9**(3/4) (1991), 365–386.

[38] A. Polleres, C. Feier and A. Harth, Rules with Contextually Scoped Negation, in: *The Semantic Web: Research and Applications, 3rd European Semantic Web Conference, ESWC 2006, Budva, Montenegro, June 11-14, 2006, Proceedings*, 2006, pp. 332–347.

[39] C. Lutz, I. Seylan and F. Wolter, Ontology-Based Data Access with Closed Predicates is Inherently Intractable(Sometimes), in: *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, 2013, pp. 1024–1030.

[40] A. Analyti, G. Antoniou, C.V. Damásio and G. Wagner, Extended RDF as a Semantic Foundation of Rule Markup Languages, *J. Artif. Intell. Res. (JAIR)* **32** (2008), 37–94.

[41] M. Schmidt, T. Hornung, M. Meier, C. Pinkel and G. Lausen, SP^{2}Bench: A SPARQL Performance Benchmark, in: *Semantic Web Information Management - A Model-Based Perspective*, 2009.

[42] C. Bizer and A. Schultz, The Berlin SPARQL Benchmark, *Int. J. Semantic Web Inf. Syst.* **5**(2) (2009), 1–24.

[43] M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte and T. Tran, FedBench: A Benchmark Suite for Federated Semantic Data Query Processing, in: *ISWC*, 2011.

[44] C. Gutierrez, D. Hernández, A. Hogan and A. 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. http://ceur-ws.org/Vol-1644/paper13.pdf.

[45] M. Arenas and J. Pérez, Querying semantic web data with SPARQL, in: *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, 2011, pp. 305–316.

[46] M. Arenas and M. Ugarte, Designing a Query Language for RDF: Marrying Open and Closed Worlds, *ACM Trans. Database Syst.* **42**(4) (2017), 21–12146.

[47] S. Razniewski, F.M. Suchanek and W. Nutt, But What Do We Actually Know?, in: *Proceedings of the 5th Workshop on Automated Knowledge Base Construction, AKBC@NAACL-HLT 2016, San Diego, CA, USA, June 17, 2016*, 2016, pp. 40–44.

[48] L. Galárraga, S. Razniewski, A. Amarilli and F.M. Suchanek, Predicting Completeness in Knowledge Bases, in: *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, WSDM 2017, Cambridge, United Kingdom, February 6-10, 2017*, 2017, pp. 375–383.

[49] P. Mirza, S. Razniewski and W. Nutt, Expanding Wikidata's Parenthood Information by 178%, or How To Mine Relation Cardinality Information, in: *Proceedings of the ISWC 2016 Posters & Demonstrations Track co-located with 15th International Semantic Web Conference (ISWC 2016), Kobe, Japan, October 19, 2016.*, 2016.

[50] P. Mirza, S. Razniewski, F. Darari and G. Weikum, Cardinal Virtues: Extracting Relation Cardinalities from Text, in: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 2: Short Papers*, 2017, pp. 347–351.

[51] R.E. Prasojo, F. Darari, S. Razniewski and W. 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.

[52] F. Darari, R.E. Prasojo and W. 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), Bethlehem, PA, USA, October 11, 2015.*, 2015.

[53] A. Letelier, J. Pérez, R. Pichler and S. Skritek, Static analysis and optimization of semantic web queries, in: *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, 2012, pp. 89–100.

[54] A. Klug, On conjunctive queries containing inequalities, *Journal of the ACM (JACM)* **35**(1) (1988), 146–160.

[55] semantic-web@w3.org Mail Archives: a blank node issue, Accessed: 2017-01-15 from https://lists.w3.org/Archives/Public/semantic-web/2011Mar/0017.html.

[56] Richard Cyganiak: Blank nodes considered harmful, Accessed: 2017-01-15 from http://richard.cyganiak.de/blog/2011/03/blank-nodes-considered-harmful/.

[57] T. Heath and C. Bizer, *Linked Data: Evolving the Web into a Global Data Space*, Synthesis Lectures on the Semantic Web: Theory and Technology, Morgan & Claypool, 2011.

[58] F. Erxleben, M. Günther, M. Krötzsch, J. Mendez and D. 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*, 2014, pp. 50–65.

[59] A. Hogan, M. Arenas, A. Mallea and A. Polleres, Everything you always wanted to know about blank nodes, *J. Web Sem.* **27** (2014), 42–69.

[60] N. Noy and A. Rector (eds), *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/.

[61] R. Cyganiak, D. Wood and M. Lanthaler (eds), *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/.

[62] A. 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*, 2015, pp. 430–440.

[63] D. 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/.

[64] S. Muñoz, J. Pérez and C. Gutierrez, Simple and Efficient Minimal RDFS, *J. Web Sem.* **7**(3) (2009), 220–234.
[65] A. Polleres, A. Hogan, R. Delbru and J. 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*, 2013, pp. 91–149.
[66] A. Motro and I. Rakov, Flexible Query Answering Systems, T. Andreasen, H. Christiansen and H.L. Larsen, eds, Kluwer Academic Publishers, Norwell, MA, USA, 1997, pp. 1–21, Chap. Not all answers are equally good: estimating the quality of database answers. http://dl.acm.org/citation.cfm?id=285506.285508.

Appendix A. Proofs of Section 3

Proposition 1. *Let $\mathcal{C}$ be a set of completeness statements, G a graph, and P a BGP. Then the following are equivalent:*

1. $\mathcal{C}, G \models \mathit{Compl}(P)$;
2. *for every mapping μ such that $\mathit{dom}(\mu) = \mathit{var}(P)$ and $(G, G \cup \mu P) \models \mathcal{C}$, it is the case that $\mu P \subseteq G$.*

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

Take the extension pair $(G, G \cup \mu P)$. By assumption, we have that $(G, G \cup \mu P) \models \mathcal{C}$. We will show that $(G, G \cup \mu P) \not\models \mathit{Compl}(P)$. Again, by assumption we have that $\mu P \not\subseteq G$. This means that $\mu \notin [\![P]\!]_G$, as opposed to the obvious fact that $\mu \in [\![P]\!]_{G \cup \mu P}$. This implies that $(G, G \cup \mu P) \not\models \mathit{Compl}(P)$. Therefore, $\mathcal{C}, G \not\models \mathit{Compl}(P)$ 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 $(G, G \cup \mu P) \models \mathcal{C}$, it is the case $\mu P \subseteq G$. We want to show that $\mathcal{C}, G \models \mathit{Compl}(P)$. Take an extension pair (G, G') such that $(G, G') \models \mathcal{C}$. 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 that $\mu P \subseteq G$. By the assumption that $(G, G') \models \mathcal{C}$ and the semantics of the $T_\mathcal{C}$ operator, we have that $T_\mathcal{C}(G') \subseteq G$. From this and $\mu P \subseteq G'$ (and also $G \subseteq G'$ by the definition of an extension pair), it holds that $T_\mathcal{C}(G \cup \mu P) \subseteq T_\mathcal{C}(G') \subseteq G$. Therefore, it is the case that $(G, G \cup \mu P) \models \mathcal{C}$. By assumption, we have that $\mu P \subseteq G$. Since μ was arbitrary, we can therefore conclude that $[\![P]\!]_{G'} \subseteq [\![P]\!]_G$. □

Appendix B. Proofs of Section 4

Proposition 3 (Equivalent Partial Grounding)**.** *Let $\mathcal{C}$ be a set of completeness statements, G a graph, and (P, ν) a partially mapped BGP. Then*

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

Proof. Take any G' such that $(G,G') \models \mathcal{C}$. We want to show that

$$[\![(P,\nu)]\!]_{G'} = \bigcup_{(\mu P,\nu\cup\mu)\in epg((P,\nu),\mathcal{C},G)} [\![(\mu P,\nu\cup\mu)]\!]_{G'}.$$

Since $dom(\nu)\cap var(P) = \emptyset$ by the construction of a partially mapped BGP, it is sufficient to show that

$$[\![(P,\mu_\emptyset)]\!]_{G'} = \bigcup_{(\mu P,\mu)\in epg((P,\mu_\emptyset),\mathcal{C},G)} [\![(\mu P,\mu)]\!]_{G'}.$$

By the construction of the *epg* operator, this corresponds to showing that

$$[\![(P,\mu_\emptyset)]\!]_{G'} = \bigcup_{\mu\in[\![cruc_{\mathcal{C},G}(P)]\!]_G} [\![(\mu P,\mu)]\!]_{G'}.$$

Recall that the crucial part of P is complete wrt. $\mathcal{C}$ and G, that is, $\mathcal{C},G \models Compl(cruc_{\mathcal{C},G}(P))$. This implies that $[\![cruc_{\mathcal{C},G}(P)]\!]_{G'} = [\![cruc_{\mathcal{C},G}(P)]\!]_G$. Therefore, it is the case that

$$\bigcup_{\mu\in[\![cruc_{\mathcal{C},G}(P)]\!]_{G'}} [\![(\mu P,\mu)]\!]_{G'} = \bigcup_{\mu\in[\![cruc_{\mathcal{C},G}(P)]\!]_G} [\![(\mu P,\mu)]\!]_{G'}.$$

By construction, we have that $cruc_{\mathcal{C},G}(P)\subseteq P$. Therefore, by the semantics of evaluating partially mapped BGPs, $[\![(P,\mu_\emptyset)]\!]_{G'} = \bigcup_{\mu\in[\![cruc_{\mathcal{C},G}(P)]\!]_{G'}} [\![(\mu P,\mu)]\!]_{G'}$. Thus, we conclude that

$$\begin{aligned}[\![(P,\mu_\emptyset)]\!]_{G'} &= \bigcup_{\mu\in[\![cruc_{\mathcal{C},G}(P)]\!]_{G'}} [\![(\mu P,\mu)]\!]_{G'} \\ &= \bigcup_{\mu\in[\![cruc_{\mathcal{C},G}(P)]\!]_G} [\![(\mu P,\mu)]\!]_{G'}.\end{aligned}$$

□

Lemma 1 (Completeness Entailment of Saturated BGPs)**.** *Let P be a BGP, $\mathcal{C}$ a set of completeness statements, and G a graph. Suppose P is saturated wrt. $\mathcal{C}$ and G. Then:*

$$\mathcal{C},G \models Compl(P) \quad \textit{iff} \quad P\subseteq G.$$

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

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

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

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

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

Since both components are in G, we have that $T_{\mathcal{C}}(G\cup\tilde{P})\subseteq G$, and therefore $(G,G\cup\tilde{P})\models\mathcal{C}$.

($\Leftarrow$) Assume $P\subseteq G$. This means that P is ground (i.e., has no variables). Therefore, it is the case that for all extension pairs (G,G'), the equation $[\![P]\!]_{G'} = [\![P]\!]_G = \{\mu_\emptyset\}$ holds, implying $(G,G')\models Compl(P)$. By definition, $\mathcal{C},G\models Compl(P)$ holds if for all $(G,G')\models\mathcal{C}$, we have $(G,G')\models Compl(P)$. Hence, $\mathcal{C},G\models Compl(P)$ holds since $(G,G')\models Compl(P)$ even for all possible extension pairs (G,G'). □

Theorem 1 (Completeness Entailment Check)**.** *Let P be a BGP, $\mathcal{C}$ a set of completeness statements, and G a graph. Then the following are equivalent:*

1. $\mathcal{C},G\models Compl(P)$;
2. $\mu P\subseteq G$, *for all* $\mu\in sat(P,\mathcal{C},G)$.

Proof. ($\Rightarrow$) We prove the contrapositive. Assume there exists a mapping $\mu \in sat(P,\mathcal{C},G)$ such that $\mu P \not\subseteq G$. From Proposition 4, we have that μP is saturated wrt. $\mathcal{C}$ and G. From Lemma 1, it is the case $\mathcal{C},G \not\models Compl(\mu P)$.

From Proposition 4, we have that $\{(P,\mu_\emptyset)\} \equiv_{\mathcal{C},G} \{(\nu P,\nu) \mid \nu \in sat(P,\mathcal{C},G)\}$. Note that by construction, each mapping in $sat(P,\mathcal{C},G)$ is incomparable to the others, in the sense that, every mapping is different. Since $\mathcal{C},G \not\models Compl(\mu P)$, we have the extension pair $(G, G \cup \mu P)$ as a counterexample for $\mathcal{C},G \models Compl(P)$.

($\Leftarrow$) By the second claim of Proposition 4, we have that μP is saturated wrt. $\mathcal{C}$ and G for each $\mu \in sat(P,\mathcal{C},G)$. Thus, from the right-hand side of Theorem 1 and Lemma 1, we have that $\mathcal{C},G \models Compl(\mu P)$ for each $\mu \in sat(P,\mathcal{C},G)$. Therefore, by the first claim of Proposition 4, we have that $\mathcal{C},G \models Compl(P)$. □

Proposition 5. *Deciding the entailment $\mathcal{C},G \models Compl(P)$, given a set $\mathcal{C}$ of completeness statements, a graph G, and a BGP P, is Π_2^P-complete.*

Proof. We prove membership in Π_2^P by showing that non-entailment is in Σ_2^P. This follows from Proposition 1, which can be read as saying that $\mathcal{C},G \not\models Compl(P)$ iff there exists a mapping μ with $dom(\mu) = var(P)$ and $\mu P \not\subseteq G$ such that

$$(G, G \cup \mu P) \models \mathcal{C}.$$

We observe that $(G, G \cup \mu P) \not\models \mathcal{C}$ iff there is a statement $Compl(P') \in \mathcal{C}$ and a mapping ν such that $\nu P' \subseteq G \cup \mu P$ and $\nu P' \not\subseteq G$. Thus, non-entailment can be checked by guessing μ in polynomial time and then performing a CoNP-check.

Next, we prove the hardness by reduction from the validity of a $\forall\exists$3SAT formula. The general shape of a formula is as follows:

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

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

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

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[31]

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

and the completeness statement

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

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

Next, we define

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

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.

Now, we define

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

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

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

to denote the seven possible satisfying value combinations for a clause. Our BGP P that 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 $v(p_i) = ?p_i$ and

[31] 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`.

the negative literal $\neg p_i$ as the variable $\upsilon(\neg p_i) = ?\overline{p}_i$. Given a clause $\gamma_i = l_{i1} \vee l_{i2} \vee l_{i3}$, the operator $tp(\gamma_i)$ maps γ_i to $(\upsilon(l_{i1}), \upsilon(l_{i2}), \upsilon(l_{i3}))$. 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:

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

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 $\mathcal{C} = \{ C_\forall, C_\psi \}$ 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 $epg((P,\mu_\emptyset),\mathcal{C},G)$ as in Algorithm 1 which involves the *cruc* operator. Accordingly, $cruc_{\mathcal{C},G}(P) = P \cap \tilde{id}^{-1}(T_\mathcal{C}(\tilde{P} \cup G))$ by definition. By construction, the statement $C_\forall$ captures the $(?x_i, varg, ?c_{x_i})$ part of the BGP P where $x_i \in vars_\forall$. Thus, by the construction of G, it is the case that $epg((P,\mu_\emptyset),\mathcal{C},G)$ 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 $epg((P,\mu_\emptyset),\mathcal{C},G)$, it is either $epg((\mu P,\mu),\mathcal{C},G) = \emptyset$ or $epg((\mu P,\mu),\mathcal{C},G) = \{\, (\mu P, \mu) \,\}$. Let us see what this means.

By construction, the former case happens whenever $T_\mathcal{C}(\widetilde{\mu P} \cup G) = \widetilde{\mu P}$ holds, from the fact that $[\![\mu P]\!]_G = \emptyset$. Furthermore, it is the case that $T_\mathcal{C}(\widetilde{\mu P} \cup G) = \widetilde{\mu P}$ iff there is a mapping ν from the encoding $?y_i$ of the existentially quantified variables in P_ψ such that $\nu(\mu P_\psi) \subseteq P_{true}$. 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 $(\mu P,\mu) \in epg((P,\mu_\emptyset),\mathcal{C},G)$, from Proposition 3 we can conclude that $(P,\mu_\emptyset) \equiv_{\mathcal{C},G} \emptyset$, and therefore $\mathcal{C}, G \models Compl(P)$. 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 $T_\mathcal{C}(\widetilde{\mu P} \cup G) \neq \widetilde{\mu P}$, since there is no mapping ν from the encoding $?y_i$ of the existentially quantified variables in P_ψ such that $\nu(\mu P_\psi) \subseteq P_{true}$. 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 $\widetilde{\mu P} \not\subseteq G$. From Lemma 1 and Proposition 3, we conclude that $\mathcal{C}, G \not\models Compl(P)$.

□

Proposition 6. *Let G be a graph. Deciding the entailment $\mathcal{C}, G \models Compl(P)$, given a set $\mathcal{C}$ of completeness statements and a BGP P, is in Π_2^P. There is a graph for which the problem is Π_2^P-complete.*

Proof. The membership follows immediately from Proposition 5, while the existence of a hard case follows from the reduction proof of that proposition, in which the graph is fixed. □

Proposition 7. *Let P be a BGP. Deciding the entailment $\mathcal{C}, G \models Compl(P)$, given a set $\mathcal{C}$ of completeness statements and a graph G, is in NP. There is a BGP for which the problem is NP-complete. This still holds when the graph is fixed.*

Proof. The membership relies on Algorithm 1 and Theorem 1. 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 the completeness statements are not fixed).

In the hardness proof we will see that the hardness follows even when the graph is fixed. The proof for NP-hardness is by means of reduction from the 3-colorability problem of classical graphs, which is known to be NP-hard. We encode the problem graph $G_p = (V, E)$, i.e., the classical graph we want to check whether it is 3-colorable, 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_{col} be:

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

Next, we create the completeness statement

$$C_p = Compl(triples(G_p) \cup P_{col}).$$

Let G be the empty graph. Then, the following claim holds:

The problem graph G_p is 3-colorable iff

$$\{ C_p \}, G \models Compl(P_{col}).$$

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. From the mapping μ, we can thus create a corresponding mapping ν such that the associated variables of the vertices in G_p are mapped to the same color as in the mapping μ. By construction of the statement C_p, it is the case that the mapping ν is a witness mapping for Q_{C_p} to derive all the triples in P_{col}, hence ensuring the completeness of P_{col}.

($\Leftarrow$) We will prove the 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. Consider the an extension pair (G, G'), where G' is the color graph $\{ (r, edge, g), \ldots, (b, edge, g) \}$. From the construction of C_p, it is the case that $(G, G') \models \{ C_p \}$ but $[\![P_{col}]\!]_G \neq [\![P_{col}]\!]_{G'}$. Thus, $\{ C_p \}, G \not\models Compl(P_{col})$. □

Proposition 8. *Let $\mathcal{C}$ be a set of completeness statements. Deciding the entailment $\mathcal{C}, G \models Compl(P)$, given a graph G and a BGP P, is in CoNP. There is a set $\mathcal{C}$ for which the problem is CoNP-complete. This still holds when the graph is fixed.*

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

- $(G, G') \models \mathcal{C}$, 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 $\mathcal{C}$ is now fixed) the entailment $(G, G \cup \mu P) \models \mathcal{C}$. If it holds, then $\mathcal{C}, G \not\models Compl(P)$ by the counterexample $G' = G \cup \mu P$.

In the hardness proof we will see that the hardness follows even when the graph is fixed. The proof for CoNP-hardness is by means of a reduction from the 3-uncolorability problem of classical graphs. We encode the problem graph $G_p = (V, E)$, i.e., the classical graph for which we want to check whether it is 3-uncolorable, 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:

$$triples(G_p) \cup \{ (c_1, c_2, c_3) \}$$

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 completeness statement

$$C = Compl((?x, edge, ?y)).$$

Then, the following claim holds:

The problem graph G_p is 3-uncolorable iff

$$\{ C \}, G \models Compl(P).$$

Proof of the claim:

($\Rightarrow$) The proof relies on Algorithm 1 and Theorem 1. First, observe that 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. Assume G_p is 3-uncolorable. As G_p is 3-uncolorable, 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 an 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 $\{ C \}, G \models Compl(P)$.

($\Leftarrow$) We will prove the contrapositive. First, observe that by construction, the part $triples(G_p)$ of the BGP P can be grounded completely due to the statement C. 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. 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 (c_1, c_2, c_3). Note that the triple pattern consists only of constants, so the mapping application has no effect. Now we have to check the completeness of (c_1, c_2, c_3). As no completeness statements can be evaluated over that remaining part, it is then the case that we are already saturated for (c_1, c_2, c_3). By Theorem 1, the BGP P can be guaranteed to be complete iff all saturated instantiations wrt. $\{ C \}$ are in G. However, clearly (c_1, c_2, c_3) is not in G. Thus, we have that $\{ C \}, G \not\models Compl(P)$. □

Proposition 9. *Let $\mathcal{C}$ be a set of completeness statements and P be a BGP. Deciding the entailment $\mathcal{C}, G \models Compl(P)$, given a graph G, is in PTIME.*

Proof. The proof relies on Algorithm 1 and Theorem 1. 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 result is therefore bounded polynomially. Moreover, now that the completeness statements are fixed, the crucial part can then be found in PTIME. Hence, the overall procedure can be executed in PTIME. □

Appendix C. Proofs of Section 5

Theorem 2. (ANSWER SOUNDNESS) *Let G be a graph, $\mathcal{C}$ a set of completeness statements, P a graph pattern, and $\mu \in [\![P]\!]_G$ a mapping. Then the following are equivalent:*

1. $\mathcal{C}, G \models Sound(\mu, P)$;
2. $\mathcal{C}, G \models Compl(\mu P_i)$, *for all* $P_i \in P^-$.

Proof. ($\Leftarrow$) Let $\mu \in [\![P]\!]_G$ be a mapping. Suppose that for all $P_i \in P^-$, we have $\mathcal{C}, G \models Compl(\mu P_i)$. Take an extension pair (G, G') satisfying $\mathcal{C}$. 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 $\mathcal{C}, G \models Compl(\mu P_i)$. 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 prove the contrapositive. Suppose there is a BGP $P_w \in P^-$ ('w' for witness) such that $\mathcal{C}, G \not\models Compl(\mu P_w)$. We will show that $\mathcal{C}, G \not\models Sound(\mu, P)$. Since it is the case that $\mathcal{C}, G \not\models Compl(\mu P_w)$, there must be a mapping ν such that: (*i*) $dom(\nu) = var(\mu P_w)$; (*ii*) $(G, G \cup \nu\mu P_w) \models \mathcal{C}$; and (*iii*) $\nu\mu P_w \not\subseteq G$. 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 $(G, G \cup \nu\mu P_w) \not\models Sound(\mu, P)$. 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, $(G, G \cup \nu\mu P_w) \not\models Sound(\mu, P)$. □

Proposition 10. *For a set $\mathcal{C}$ of completeness statements and BGPs P and P', it is the case that*

$$\mathcal{C} \models Compl(P \mid P') \quad \textit{iff} \quad \tilde{P} \subseteq T_{\mathcal{C}}(\tilde{P} \cup \tilde{P'}).$$

Proof. ($\Rightarrow$) Suppose that $\mathcal{C} \models Compl(P \mid P')$. By definition of the entailment, for all $(G, G') \models \mathcal{C}$, the inclusion $[\![(var(P), P \cup P')]\!]_{G'} \subseteq_s [\![P]\!]_G$ holds. Consider the extension pair (G, G') where $G = T_{\mathcal{C}}(\tilde{P} \cup \tilde{P'})$ and $G' = \tilde{P} \cup \tilde{P'}$. By construction, $(G, G') \models \mathcal{C}$ holds. From our assumption, it follows that $[\![(var(P), P \cup P')]\!]_{G'} \subseteq_s [\![P]\!]_G$. We define the operator $\pi_W(\mu)$ by projecting the mapping μ to the variables in W. 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 $\pi_{var(P)}(\tilde{id})P = \tilde{P} \subseteq G = T_{\mathcal{C}}(\tilde{P} \cup \tilde{P'})$.

($\Leftarrow$) Assume $\tilde{P} \subseteq T_{\mathcal{C}}(\tilde{P} \cup \tilde{P'})$. 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 $\mathcal{C} \models Compl(P \mid P')$. □

Lemma 2. *Let $\mathcal{C}$ be a set of completeness statements and P a graph pattern. Then $\mathcal{C} \models Sound(P)$ provided that $\mathcal{C} \models Compl(P_i \mid P^+)$ for all $P_i \in P^-$.*

Proof. Assume that for all $P_i \in P^-$, it is the case that $\mathcal{C} \models \mathit{Compl}(P_i \mid P^+)$. Take any extension pair $(G, G') \models \mathcal{C}$ 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 $\mathcal{C} \models \mathit{Compl}(P_i \mid P^+)$ and $(G, G') \models \mathcal{C}$, 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 $\mu \in [\![P]\!]_{G'}$. □

Theorem 3. (PATTERN SOUNDNESS) *Let $\mathcal{C}$ be a set of completeness statements and P a graph pattern in NRF. Then the following are equivalent:*

1. $\mathcal{C} \models \mathit{Sound}(P)$;
2. $\mathcal{C} \models \mathit{Compl}(P_i \mid P^+)$ *for all* $P_i \in P^-$.

Proof. ($\Leftarrow$) This is a direct consequence of Lemma 2. ($\Rightarrow$) We give a proof by contrapositive. Suppose there is a BGP $P_w \in P^-$ ('w' for witness) such that $\mathcal{C} \not\models \mathit{Compl}(P_w \mid P^+)$. By Proposition 10, it is the case that $\tilde{P}_w \not\subseteq T_{\mathcal{C}}(\tilde{P}_w \cup \tilde{P}^+)$. Let us prove that for the extension pair $(G, G') = (\tilde{P}^+ \cup T_{\mathcal{C}}(\tilde{P}_w \cup \tilde{P}^+), \tilde{P}_w \cup \tilde{P}^+)$, it is the case that $(G, G') \models \mathcal{C}$, but $(G, G') \not\models \mathit{Sound}(P)$.

By the definition of $T_{\mathcal{C}}$, it holds that $(G, G') \models \mathcal{C}$. 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 $\tilde{id} \in [\![P]\!]_{\tilde{P}^+ \cup T_{\mathcal{C}}(\tilde{P}_w \cup \tilde{P}^+)} = [\![P]\!]_G$.

By construction, $\tilde{id} \in [\![P^+]\!]_{\tilde{P}^+ \cup T_{\mathcal{C}}(\tilde{P}_w \cup \tilde{P}^+)}$. Thus, it is left to show that for every BGP $P_i \in P^-$, it is the case $[\![\tilde{id} P_i]\!]_{\tilde{P}^+ \cup T_{\mathcal{C}}(\tilde{P}_w \cup \tilde{P}^+)} = \emptyset$. Since, ue to the non-redundancy property of NRF graph patterns, different negated patterns do not contain one another, there is no negated pattern $P_j \neq P_w$ such that:

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

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

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

However, this follows from the minimality of negated patterns in 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 $(G, G') \models \mathit{Sound}(P)$. □

Proposition 11. *Negation-similar graph patterns are equivalent.*

Proof. Suppose that P and $\bar{P}$ are negation-similar. We show that $P \sqsubseteq \bar{P}$. The converse statement follows by a symmetric argument.

Let G be a graph and let $\mu \in [\![P]\!]_G$. We want to show that $\mu \in [\![\bar{P}]\!]_G$. Clearly, $\mu \in [\![P^+]\!]_G$ and therefore also $\mu \in [\![\bar{P}^+]\!]_G$. To show that $\mu \in [\![\bar{P}]\!]_G$, we must ensure that $[\![\mu \bar{P}_j]\!]_G = \emptyset$ for all $\bar{P}_j \in \bar{P}^-$. By assumption, for $\bar{P}_j \in \bar{P}^-$ there exists a $P_i \in P^-$ such that $(\mathit{var}(\bar{P}^+), \bar{P}^+ \cup \bar{P}_j) \sqsubseteq_s (\mathit{var}(P^+), P^+ \cup P_i)$. Since $\mu \in [\![P]\!]_G$, we have $[\![\mu P_i]\!]_G = \emptyset$, which implies that $[\![\mu \bar{P}_j]\!]_G \subseteq [\![\mu P_i]\!]_G = \emptyset$. This completes the proof. □