Semantic Web 0 (0) 1
IOS Press



# Ensuring the Completeness and Soundness of SPARQL Queries Using Completeness Statements about RDF Data Sources[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 to bridge the gap between RDF and SPARQL. We first develop a technique to check query completeness based on RDF data with completeness information. For queries with negation, we approach the problem of query soundness checking. We provide a formalization and characterize the soundness problem via a reduction to the completeness problem. We further develop efficient methods for completeness checking, and conduct experimental evaluations based on Wikidata 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 January 2017, the LOD Cloud[2] has recorded more than 1,100 RDF data sources, covering a wide range of application domains from government to life sciences. RDF follows the open-world assumption (OWA), which assumes 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 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 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.

[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 March 21, 2017)
[4] See, e.g., http://www.imdb.com/title/tt0105236/fullcredits
[5] https://europa.eu/european-union/about-eu/history/


Fig. 1. Wikidata is actually 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 around 14,500 pages with the keywords 'complete list of' and 'list is complete'; IMDb offers about 37,000 editor-verified (natural language) statements about the completeness of cast and crew;[7] and OpenStreetMap has around 2,300 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.

*Query Completeness* When data sources are enriched with completeness information, the question naturally arises 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] characterized such reasoning, that is, the checking whether queries can be guaranteed to be complete by completeness statements. Nonetheless, their approach is limited in the sense that the specifics of the 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 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] 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.

*Query Soundness* One might wonder whether the information about completeness 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. The meaning of such queries on the Semantic Web has always been dubious (see, e.g., W3C mailing list discussions in [6]

[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

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

and [7]). 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 of 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 and 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 present paper significantly extends the previous work in the following ways:

1. we formulate the soundness problem for SPARQL queries with negation in the presence of completeness information, and characterize it via a reduction to completeness checking;
2. we identify the bottlenecks of the completeness reasoning techniques from [1] and develop implementation techniques that scale to realistic problem sizes;
3. we provide experimental evaluations based on Wikidata, a real-world data source, that validate the feasibility of our approach; and
4. we provide a comprehensive complexity analysis of the completeness entailment problem, include the proofs of all theorems as well as more recent related work, and improve the presentation of the theoretical parts.

A preliminary result of the soundness problem appeared in a poster by Darari et al. [8], in which only a partial characterization was given.

*Organization* The rest of the article is organized as follows. Section 2 provides some background about RDF and SPARQL, and completeness statements. Section 3 motivates and formalizes the problem of query completeness and query soundness. In Section 4, we first introduce formal notions, and then present an algorithm for completeness entailment checking using those notions, and 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 optimization 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.

### 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* $G$ is represented as 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 query language for RDF is SPARQL [5]. At the core of SPARQL lie triple patterns, which resemble triples, except that in each position also vari-

ables are allowed. A *basic graph pattern* (BGP) consists of a set of triple patterns. A mapping $\mu$ is a partial function $\mu\colon V \to I \cup L$. We define the mapping with the empty domain as the *empty mapping* $\mu_\emptyset$. Given a BGP $P$, $\mu P$ denotes the BGP obtained by replacing variables in $P$ with terms according to $\mu$. Evaluating $P$ over a graph $G$ gives the set of mappings $[\![P]\!]_G = \{\mu \mid \mu P \subseteq G \text{ and } dom(\mu) = var(P)\}$. For a BGP $P$, we define the *freeze mapping* $\tilde{id}$ as mapping each variable $?v$ in $P$ to a fresh IRI $\tilde{v}$. From such a mapping, we construct the *prototypical graph* $\tilde{P} := \tilde{id}\,P$ to represent any possible graph that can satisfy the BGP $P$.

The standard query type is a `SELECT` query, which has the abstract form $Q = (W, P)$, where $P$ is a graph pattern and $W \subseteq var(P)$. Under bag semantics for such queries, $[\![Q]\!]_G$ is obtained by projecting the mappings in $[\![P]\!]_G$ to $W$, keeping each projection as many times as there are mappings that give rise to it. In this work, we assume bag semantics for `SELECT` queries as it is the default semantics of SPARQL [5].

Given two BGPs $P_1$ and $P_2$ where $var(P_1) \subseteq var(P_2)$, a `CONSTRUCT` query has the abstract form (`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: $\{\mu \in [\![P^+]\!]_G \mid \forall P_i \in P^- . [\![\mu P_i]\!]_G = \emptyset\}$. We assume that graph patterns are consistent, that is, $[\![P]\!]_G \neq \emptyset$ for some graph $G$.

**Example 1.** Consider the query "Give the founding members of the EU" as introduced in Section 1. It can be written as: $(\{?c\}, \{(EU, founder, ?c)\})$. Consider also negated version "Give countries that are not EU founders." The graph pattern of the query can be written as: $\{(?c, a, country), \neg\exists\{(EU, founder, ?c)\}\}$. The positive part is $\{(?c, a, country)\}$, whereas the negative part is $\{\{(EU, founder, ?c)\}\}$.

### 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 1** (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 definition of completeness statements in [4].

**Definition 2** (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 $C_{eu} = Compl((EU, founder, ?f))$.[10] 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 $Q_C =$ (`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 $Q_{C_{eu}} =$ (`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 $G$.

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

**Definition 3** (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. Given a set $\mathcal{C}$ of completeness statements and a graph $G$, the *transfer operator* $T_\mathcal{C}$ maps the graph $G$ to the graph $T_\mathcal{C}(G) = \bigcup_{C \in \mathcal{C}} [\![Q_C]\!]_G$. The transfer operator evaluates over $G$ all `CONSTRUCT` queries associated to the statements in $\mathcal{C}$ and returns the union of the results. We have the following immediate characterization: for all extension pairs $(G, G')$, it is the case that $(G, G') \models \mathcal{C}$ iff $T_\mathcal{C}(G') \subseteq G$.

## 3. Motivation and Formal Framework

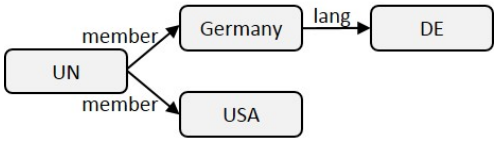
In this section, we motivate and formalize the problem of query completeness and query soundness.

### 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.

#### 3.1.1. Motivating Scenario

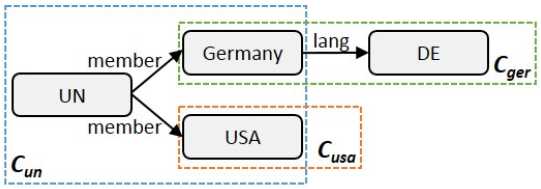
Consider the following RDF graph $G_{cou}$ about members countries of the United Nations (UN) and official languages of the members.




Consider now 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 result, 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.




Let us now consider the same graph as above, enriched with completeness information, as displayed below. The figure illustrates the set $\mathcal{C}_{cou}$ of three completeness statements:

- $C_{un} = Compl((UN, member, ?m))$, which states that the graph contains all members of the UN;[11]
- $C_{ger} = Compl((ger, lang, ?l))$, 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 have no official languages).[12]

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

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 $(usa, lang, ?l)$. 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 $(usa, lang, ?l)$ returns nothing (i.e., the USA has no official languages). In other words, for any possible

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

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

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 $Q_3$. In this case, 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. 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 4** (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, $[\![Q]\!]_{G'} \subseteq [\![Q]\!]_G$.*[13] *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 5** (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 there entail the completeness of the query $Q_0$ asking for members of the UN and their official languages.

As we follow the default of SPARQL and assume bag semantics for query evaluation, we can therefore focus on the BGPs used in the body of queries for completeness entailment. The following proposition provides an initial characterization of completeness entailment, 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, iff extending the graph with a possible BGP instantiation (by some mapping) such that the

[13] 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.

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 $\mu$ 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, iff 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).

### 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:

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

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

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

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.[14] 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

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

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 $\{?c \mapsto \tilde{c}\}$ for an IRI $\tilde{c}$. Consider also an arbitrary extension $G'$ of $G$ such that $(G, G') \models \mathcal{C}_f$. To show that $\{?c \mapsto \tilde{c}\}$ is sound, we must make sure that neither over $G$ nor over $G'$ does $\tilde{c}$ have `en` as an official language and is $\tilde{c}$ 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 languages of $\tilde{c}$. The fact that $\tilde{c}$ 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 $\tilde{c}$ is returned over $G$ means that $\tilde{c}$ 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 $\{?c \mapsto \tilde{c}\}$ 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

Let us first define what soundness of an answer means. Consider a graph pattern $P$, a mapping $\mu$, and an extension pair $(G, G')$. We say that $(G, G')$ satisfies the soundness of $\mu$ for $P$, written $Sound(\mu, P)$ if, whenever $\mu \in [\![P]\!]_G$, then also $\mu \in [\![P]\!]_{G'}$. As for $\mu \notin [\![P]\!]_G$ it is trivial that $(G, G') \models Sound(\mu, P)$, we are only interested in the soundness of answers occurring in $[\![P]\!]_G$. 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* $\mu$ *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 *usa* is a sound answer while *sgp* is not, thus $\mathcal{C}_l, G_l \models Sound(\{?c \mapsto usa\}, P_l)$, whereas $\mathcal{C}_l, G_l \not\models Sound(\{?c \mapsto sgp\}, P_l)$.

In defining pattern soundness, as opposed to answer soundness, we are concerned with a graph pattern as a whole, and abstract from any specific answers of the pattern. For a graph pattern $P$, we express that $P$ *is sound* by writing $Sound(P)$. Given an extension pair $(G, G')$, we say the $P$ *is sound over* $(G, G')$, written $(G, G') \models Sound(P)$, if $[\![P]\!]_G \subseteq [\![P]\!]_{G'}$. Given a set $\mathcal{C}$ of completeness statements and a graph pattern $P$, we say that $\mathcal{C}$ *entails the soundness of* $P$, written as $\mathcal{C} \models Sound(P)$, 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 $\mathcal{C}_f \models Sound(P_f)$.

It follows immediately from the definitions that all answers of a sound pattern 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 $\mu$.*

## 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. Let $P$ be a BGP and $\mu$ be a mapping such that $dom(\mu) \cap var(P) = \emptyset$. We define the pair $(P, \mu)$ as a *partially mapped BGP*, which is a BGP with a stored mapping. Over a graph $G$, the evaluation of $(P, \mu)$ is defined as $[\![(P, \mu)]\!]_G = \{\mu \cup \nu \mid \nu \in [\![P]\!]_G\}$. It is easy to see that $P \equiv (P, \emptyset)$. Furthermore, we define the evaluation of a set of partially mapped BGPs over a graph $G$ as the union of evaluating each of them over $G$.

**Example 2.** 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 want to formalize the equivalence between partially mapped BGPs wrt. a set $\mathcal{C}$ of completeness statements and a graph $G$. We need this notion to ensure the equivalence of the BGP instantiations that resulted from the evaluation of complete BGP parts.

**Definition 6** (Equivalence under $\mathcal{C}$ and $G$)**.** *Let $(P,\mu)$ and $(P',\nu)$ be partially mapped BGPs, $\mathcal{C}$ be a set of completeness statements, and $G$ be a graph. We define that $(P,\mu)$ is* equivalent *to $(P',\nu)$ 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 3.** Consider the queries in our motivating scenario. It is the case that $\{ (P_0,\emptyset) \} \equiv_{\mathcal{C}_{cou},G_{cou}} \{ (P_1,\{ ?m \mapsto ger \}),(P_2,\{ ?m \mapsto usa \}) \} \equiv_{\mathcal{C}_{cou},G_{cou}} \{ (P_3,\{ ?m \mapsto ger, ?l \mapsto de \}) \}$.

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, while the rest to the graph. For this reason, we define

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

as the *crucial part of $P$ wrt. $\mathcal{C}$ and $G$*. It is the case that 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 4.** Consider the query $Q_0 = (W_0,P_0)$ in our motivating scenario. We have that $cruc_{\mathcal{C}_{cou},G_{cou}}(P_0) = P_0 \cap \tilde{id}^{-1}(T_{\mathcal{C}_{cou}}(\tilde{P}_0 \cup G_{cou})) = \{ (UN, member, ?m) \}$ with $\tilde{id} = \{ ?m \mapsto \tilde{m}, ?l \mapsto \tilde{l} \}$. Consequently, we can have a complete instantiation of the UN members.

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

**Definition 7** (Equivalent Partial Grounding)**.** *Let $\mathcal{C}$ be a set of completeness statements, $G$ be a graph, and $(P,\nu)$ 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 $(P,\nu)$ a partially mapped BGP. Then*

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

**Example 5.** Consider our motivating scenario. We have that:

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

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.

From these three cases and the finite number of triple patterns with variables of a BGP, it holds that the repeated applications of the *epg* operator, with the first and second cases above as the base cases, are terminating. Note that the difference between these two base cases is in the effect of their corresponding *epg* operations, as illustrated in Example 5: for the first case, the *epg* operation returns an empty set, whereas for the second case, it returns back the input partially mapped BGP. Intuitively, the first case corresponds to the non-existence of the query answer in any possible extension of the graph that satisfies the set of completeness statements (e.g., the USA's official languages case).

As for the second case, we need a different treatment. We first define that a partially mapped BGP $(P,\nu)$ 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 from the mapping in a partially mapped BGP: given a mapping $\nu$, a partially mapped BGP $(P,\nu)$ is saturated wrt. $\mathcal{C}$ and $G$ iff $(P,\nu')$ is saturated wrt. $\mathcal{C}$ and $G$ for any mapping $\nu'$. 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 $G$.

**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:*

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

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, given a set of completeness statements $\mathcal{C}$, a graph $G$, and a BGP $P$, all mappings that have two properties: each BGP instantiation of the mappings constitutes a saturated BGP wrt. $\mathcal{C}$ 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.

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

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

$\mathbf{P}_{working} \leftarrow \{ (P_{orig}, \mu_\emptyset) \}$
$\Omega \leftarrow \emptyset$
**while** $\mathbf{P}_{working} \neq \emptyset$ **do**
  $(P, \nu) \leftarrow$ `takeOne`$(\mathbf{P}_{working})$
  $\mathbf{P}_{equiv} \leftarrow \mathit{epg}((P, \nu), \mathcal{C}, G)$
  **if** $\mathbf{P}_{equiv} = \{ (P, \nu) \}$ **then**
    $\Omega \leftarrow \Omega \cup \{ \nu \}$
  **else**
    $\mathbf{P}_{working} \leftarrow \mathbf{P}_{working} \cup \mathbf{P}_{equiv}$
  **end**
**end**
**return** $\Omega$

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 $(P, \nu)$ from $\mathbf{P}_{working}$ via the method `takeOne`. Afterwards, we compute $\mathit{epg}((P, \nu), \mathcal{C}, G)$. As discussed above there might be three result cases here: *(i)* If $\mathit{epg}((P, \nu), \mathcal{C}, G) = \emptyset$, then we simply remove $(P, \nu)$ and will not consider it anymore in the later iteration; *(ii)* If $\mathit{epg}((P, \nu), \mathcal{C}, G) = \{ (P, \nu) \}$, that is, $(P, \nu)$ is saturated, then we add the mapping $\nu$ to the set $\Omega$; and *(iii)* otherwise, we add to $\mathbf{P}_{working}$ a set of partially mapped BGPs instantiated from $(P, \nu)$. We keep iterating until $\mathbf{P}_{working} = \emptyset$, and finally return the set $\Omega$.

The following proposition follows from the construction of the above 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:*

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

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

**Example 6.** Consider our motivating scenario. Then $\mathit{sat}(P_0, \mathcal{C}_{cou}, G_{cou}) = \{ \{ ?m \mapsto ger, ?l \mapsto de \} \}$. For every mapping $\mu$ in $\mathit{sat}(P_0, \mathcal{C}_{cou}, G_{cou})$, it holds that $\widetilde{\mu P_0} \subseteq G_{cou}$. Thus, by Theorem 1 the entailment $\mathcal{C}_{cou}, G_{cou} \models \mathit{Compl}(P_0)$ 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 $\mathit{sat}(P, \mathcal{C}, G)$, which we know how to compute.

#### 4.2.1. Simple Practical Optimizations

In what follows we provide two simple optimization techniques of the algorithm: early failure detection and completeness skip. More elaborate optimizations are given in 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 $(P,\nu)$ 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 $\Pi_2^P$-complete. The hardness is by reduction from the validity problem of a $\forall\exists$3SAT formula.

**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 $\Pi_2^P$-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.

First, let us fix the input graph $G$. This does not change the complexity, that is, the problem is still $\Pi_2^P$-complete. The reason is that the reduction from the validity problem of a $\forall\exists$3SAT formula can be done even with a fixed graph.

**Proposition 6.** *Deciding the entailment $\mathcal{C},G \models Compl(P)$, given a set $\mathcal{C}$ of completeness statements, a fixed graph $G$, and a BGP $P$, is $\Pi_2^P$-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.** *Deciding the entailment $\mathcal{C},G \models Compl(P)$, given a set $\mathcal{C}$ of completeness statements, a graph $G$, and a fixed BGP $P$, is NP-complete. The complexity remains the same even 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.** *Deciding the entailment $\mathcal{C},G \models Compl(P)$, given a fixed set $\mathcal{C}$ of completeness statements, a graph $G$, and a BGP $P$, is CoNP-complete. The complexity stays the same even 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.** *Deciding the entailment $\mathcal{C},G \models Compl(P)$, given a fixed set $\mathcal{C}$ of completeness statements, a graph $G$, and a fixed BGP $P$, is in PTIME.*

This result corresponds to some practical cases when queries are assumed to be of limited length[15] and hence, so are completeness statements (which are essentially also queries).

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 optimization techniques, as well as experimental evaluations of the problem.

[15] as also customary in database theory when analyzing the data complexity of query evaluation [9]

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

| *input* | | | *complexity* |
|---|---|---|---|
| $\mathcal{C}$ | $G$ | $P$ | |
| ✓ | ✓ | ✓ | $\Pi_2^p$-C |
| ✓ | × | ✓ | $\Pi_2^p$-C |
| ✓ | ✓ | × | NP-C |
| ✓ | × | × | NP-C |
| × | ✓ | ✓ | CoNP-C |
| × | × | ✓ | CoNP-C |
| × | ✓ | × | in PTIME |

## 5. Checking Query Soundness

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

### 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 7.** 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 subsection. We only need that the positive part of the pattern be monotonic, that is, a mapping remains a solution over all extensions of the graph $G$. We do not make this formal to keep the exposition simple.

*Complexity* From Theorem 2, the check whether an answer is sound wrt. a set of completeness statements and a graph can be reduced to a linear number of data-aware completeness checks (as discussed in Section 4). From this, it follows that the complexity of the answer soundness entailment problem is in $\Pi_2^p$. Moreover, the answer soundness problem is also $\Pi_2^p$-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 $[\![(var(P), P \cup P')]\!]_{G'} \subseteq_s [\![P]\!]_G$.[16] 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 Subsection 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 the case that $(G, G') \models Compl(P \mid P')$. The following proposition states that such entailment holds iff the $T_\mathcal{C}$ application over 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.

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

**Example 8.** Consider the following graph patterns:

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

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

Both return as answers mappings $\{?c \mapsto \tilde{c}\}$, where $\tilde{c}$ is a country that does not have the language `en`. The reason is that, after instantiating $?c$ with $\tilde{c}$, all three negated subpatterns are empty over a graph $G$ if and only if the triple $(\tilde{c}, lang, en)$ is not present in $G$. Consider next also the completeness statement $\mathcal{C} = \{Compl((?c, lang, en))\}$. If the triple $(\tilde{c}, lang, en)$ 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 $\{?c \mapsto \tilde{c}\}$ is also 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 statement $Compl((?c, lang, en))$ is sufficient to guarantee their soundness.

To avoid such redundancies, we propose a normal form for graph patterns, called *non-redundant form (NRF)*. A graph pattern $P$ is in NRF if it satisfies 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

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

A non-NRF graph pattern can be transformed into an equivalent NRF graph pattern with a polynomial

number of NP-checks, by repeating the containment check and redundant part removal until the two conditions above are satisfied. As graph patterns tend to be small in practice, we expect that such a transformation is feasible.

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 whether it is sound can be reduced to the check 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 9.** In the motivating scenario of pattern soundness, it holds that

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

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

*Complexity* From Theorem 3 and Proposition 10, it follows that the check whether a graph pattern is sound can be reduced to a linear number of $T_{\mathcal{C}}$ applications, which are basically evaluations of conjunctive `CONSTRUCT` queries. Hence, deciding whether a graph pattern is sound wrt. a set of completeness statements is in NP (and also NP-hard, as checking completeness can also be reduced to checking soundness). From a practical viewpoint, one may expect graph patterns of queries and BGPs of completeness statements to be short, potentially allowing for a feasible soundness check. 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 positive part of 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 10.** Consider the following boolean query, which asks whether it is impossible to right-shift any triple:

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

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

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

We assume bag semantics, thus $Q$ returns a bag of empty mappings $\{\}$ when evaluated of a graph $G$. 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 $G$. Now suppose that $Q$ returns a copy of $\{\}$ 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 $Q$ continues to return a copy of $\{\}$ 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 $\{\}$, 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 completeness 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. Via both characterizations, we can then check whether a graph pattern is sound and/or complete.

*`MINUS` Operator* SPARQL knows two operators for expressing negation, `NOT-EXISTS` and `MINUS`. While in the presentation we have focused on negation via `NOT-EXISTS`, 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.

## 6. Optimizing Completeness Checking

Up until now, we have shown how completeness entailment can be characterized, and how soundness checking amounts to completeness checking. In this section, we present techniques to optimize both data-agnostic and data-aware completeness checking and in the next section, we report on experiments with them.

Similarly to queries, which are short in many practical cases [10, 11], 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. 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 of similar topics, we introduce completeness templates. By providing a compact representation of completeness statements, completeness templates enable multiple statements to be processed simultaneously.

### 6.1. Data-agnostic Completeness Checking

By Theorem 3, pattern soundness checking can be reduced to the (data-agnostic) check whether a set of completeness statements entails a conditional completeness statement. Proposition 10 states that one can check whether $Compl(P \mid P')$ is entailed by a set of statements $\mathcal{C}$ by evaluating the union of the `CONSTRUCT` queries $Q_C$, for all $C \in \mathcal{C}$, 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., the 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 and, as we will show later, efficient way 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 much fewer than the original statements.

**Example 11.** 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:

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

Consider the query $P_f$ from Subsubsection 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 the 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 former case. 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.

### 6.2. Data-aware Completeness Checking

For the data-aware setting, reasoning needs also access to the data graph. The previous approach to optimization of data-agnostic reasoning, which leaves out statements whose terms are not among the terms of

the query, is no more applicable, since parts of the statements can now be mapped to the data graph. We present a new data structure, called completeness templates, that bundles similar statements, and a new completeness checking algorithm that works on these templates and generalizes the earlier one in Section 4.

*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,[17] or in OpenStreetMap, which uses a wiki to record the completeness of objects in different areas.[18] 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 $\Omega$ is a set of mappings from $V_\tau$ to terms (i.e., IRIs or literals). We also refer to the BGP of the completeness statement $C$ of the template $\tau$ as $P_\tau$. 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 $\Omega$. This definition naturally extends to sets of completeness templates. Note that a completeness statement $C$ can be expressed as the completeness template $(C, \emptyset, \{\mu_\emptyset\})$, where $\mu_\emptyset$ is the empty mapping.

*Template-based Transfer Operator* A key part of the algorithm for checking data-aware completeness, 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 $\tilde{P}_0 \subseteq T_\mathcal{C}(\tilde{P} \cup G)$. 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 $\tau$, we first evaluate $P_\tau$ (by treating the meta-variables like variables) over the union graph $\tilde{P} \cup G$, with the condition that at least one triple pattern in $P_\tau$ is mapped to a triple in $\tilde{P}$ (since otherwise the mapping does not contribute to $P_0$), and verify in a second step which of the resulting mappings are compatible with the instantiations of the template $\tau$. In this way, all instances of $\tau$ can be processed simultaneously.

To formalize the above idea, we first define prioritized evaluation of a BGP over a pair of graphs $(G_1, G_2)$. In such an evaluation, we consider the first graph $G_1$ as the *mandatory* and the second as the *optional* graph, which means that at least one triple pattern of the BGP is mapped to a triple 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 $[\![P]\!]_{(G_1,G_2)} = \{\mu \mid \mu \in [\![P]\!]_{G_1 \cup G_2}$ and $\mu P' \subseteq G_1$ 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 $G$.

**Example 12.** 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 template $\tau_{org} = (C, \{?org\}, \Omega)$, where

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

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

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

[17] See e.g., http://www.imdb.com/title/tt0105236/fullcredits
[18] See e.g., http://wiki.openstreetmap.org/wiki/Abingdon

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

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

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

Each set $\mathcal{T}$ of completeness templates gives rise to a *template-based transfer operator* $T_\mathcal{T}$. Given a frozen BGP $\tilde{P}$, and a graph $G$, the operator maps the pair to the set of triples

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

The above operator computes for each template $\tau$ the prioritized evaluation of the BGP $P_\tau$ over $(\tilde{P}, G)$, keeps only those mappings compatible with $\Omega$, and then takes the union. The crucial point here is that we can first evaluate the BGP of the template, and only after that we check which answers correspond to instantiations by $\Omega$. By the definition of completeness templates and the prioritized evaluation of BGPs, it is the case that $P_0$ as in Eq. (2) can alternatively be computed using $T_\mathcal{T}$, as stated in Proposition 11.

**Proposition 11.** *Given a BGP $P$, a graph $G$, and a set $\mathcal{T}$ of completeness templates, it is the case that*

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

*Partial Matching* As there can be many completeness templates, we want to rule out irrelevant ones, that is, ones that do not contribute to query completeness. Clearly, templates cannot contribute if their triple patterns do not overlap with the query (modulo variable generalization). If a template does overlap with the query, however, we say that it partially matches the query.

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

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

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

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

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

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

In practice, such a mapping can be realized by standard hashmaps, providing fast retrieval operations. Given a signature triple $(s, p, o)$, the *generalization operator* $\texttt{gen}((s, p, o))$ computes the set of all generalizations where non-predicate terms can become variables. As an illustration, the generalization of the signature triple $(usa, lang, \texttt{\_var})$ 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 templates $\mathcal{T}$, retrieves those elements of $\mathcal{T}$ that can potentially 'transfer' at least one triple in the frozen BGP $\tilde{P}$. Technically, `pmatch` maps

$P$ and $\mathcal{T}$ to the set of *partially matched* templates wrt. $P$ and $\mathcal{T}$, denoted $\texttt{pmatch}(P,\mathcal{T})$, 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 templates in $\mathcal{T}$. By the construction of the mapping $M$ and the generalization operator, it is the case that $\texttt{pmatch}(P,\mathcal{T})$ preserves $P_0$ in Eq. (2), as stated in Proposition 12.

**Proposition 12.** *Given a BGP $P$, a graph $G$, and a set $\mathcal{T}$ of completeness templates, it holds that*

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

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

## 7. Experimental Evaluation

This section reports our experimental evaluation of query completeness and query soundness reasoning.

### 7.1. Query Completeness Evaluation

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

#### 7.1.1. Experimental Setup

The reasoning program and experiment framework were implemented in Java using the Apache Jena library[19] and are available online.[20] We used the direct-statement fragment (i.e., the fragment with no qualifiers nor references) of Wikidata as our *data graph*, consisting of around 110 mio triples.[21] We chose Wikidata mainly because of its relatively large size, recent popularity, and good data quality, making it suitable for our data-aware experiment. The graph was loaded into a Jena TDB triple store.

Our *queries* were generated based on human-made, openly available queries on the Wikidata query page.[22] We extracted the BGPs of the queries and transformed the vocabulary of the queries to the direct statement vocabulary. These BGPs acted as a 'base' for generating our experiment queries: *(i)* for each base, we evaluated it over the Wikidata graph; *(ii)* we took randomly 20 of the result mappings of the base, projected on the first variable of the base;[23] and *(iii)* we generated queries by instantiating the query bases with these projected mappings. The *completeness statements* are generated in a similar way: *(i)* for each base, we evaluated it over the Wikidata graph; *(ii)* from the answer mappings, we took randomly 50% of them, projected to the first variable of the base; and *(iii)* we generated completeness statements by instantiating the base with the respective mappings as the statements' BGPs. In this setting, we also naturally represented completeness statements by completeness templates as follows: we took the base BGP as the template's BGP, and the projected mappings as the template's mappings.

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

#### 7.1.2. Results and Discussion

In the experiments, we observed the query evaluation time and completeness reasoning time from 1,160 queries, with the average query length of 2.58. There were 445,628 completeness statements generated, with the average completeness statement length of 2.43 (i.e., the number of triple patterns in the BGP of completeness statements). Furthermore, those statements were represented by 66 completeness templates, corresponding to the number of base BGPs to generate the queries. On average, query evaluation took 2.23

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

[20] http://completeness.inf.unibz.it/data-aware-completeness-experiment/

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

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

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

ms, whereas completeness reasoning took 140.09 ms, which was still relatively fast.

Table 2
Average runtime comparison of query evaluation and completeness reasoning grouped by query length, where $|Q|$ is the query length, $N_Q$ is the number of queries, $t_Q$ is the average of query evaluation time, and $t_C$ is the average of completeness reasoning time.

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

To get more detailed observations, we broke down the experiment results by query length (as shown in Table 2). There is no clear pattern for both query evaluation and completeness reasoning as it is not always the case that the longer the query gets, the longer the runtime becomes. Interestingly though, the completeness reasoning time for queries of length 1 is much faster than the others. This is likely due to smaller partial matches with templates and easier prioritized evaluation in the reasoning, in the sense that processing such queries does not even need to see the data graph whenever the corresponding templates' BGPs are also of length 1 (recall Subsection 6.2). Overall, though completeness reasoning is slower than query evaluation, it is still relatively fast (i.e., always below 700 ms). To get an idea of how long plain completeness reasoning takes (i.e., without optimizations), we chose randomly 10 queries for each query length group and measured the reasoning time. We then computed the average reasoning time with a weighting scheme that respects the query distribution as in Table 2. The average reasoning time was 15 s, which is relatively slow. A possible explanation is that for plain completeness reasoning, all the statements were applied repeatedly over the union of the frozen BGP $\tilde{P}$ and the data graph $G$. We then measured the reasoning time for those queries using only the partial matching technique, where we constructed a single completeness template for every completeness statement. In this case, the average reasoning time was 401.8 ms, as opposed to 140.09 ms, where completeness templates to represent multiple statements were additionally used. Without using templates, partial matching might still get many completeness statements that have to be individually evaluated over $\tilde{P} \cup G$, as opposed to the template's simultaneous processing. This shows that both optimization techniques, that is, completeness templates and partial matching, may help speed up the reasoning.

### 7.2. Query Soundness Evaluation

From the characterizations in Section 5, we are able to check query soundness by reducing it to query completeness checks. For this reason, we can reuse the optimization techniques of completeness reasoning as described in Section 6. More specifically, we reuse the term-relevance technique for optimizing pattern soundness checking, and the completeness templates and partial matching techniques for optimizing answer soundness checking. In this subsection, we analyze how soundness reasoning behaves in a realistic scenario, in particular: how feasible it is to perform soundness reasoning, how much speed-up can be gained with the optimization techniques, and how does pattern soundness checking compare to answer soundness checking. This subsection reports our experimental evaluation based on Wikidata. First, we describe our experimental setup, and then discuss the results of the experiments.

#### 7.2.1. Experimental Setup

The reasoning program and experiment framework were implemented in Java using the Apache Jena library, and are available online.[24] As it was the case for the data-aware completeness experiment in Subsection 7.1, we also used the direct statements fragment of Wikidata as our *data graph*. The graph was loaded into a Jena TDB triple store.

*Queries* Similarly to the query completeness experiment, we took Wikidata sample queries, and used them as templates to generate queries, this time with negation.[25] We extracted the BGPs of the queries and transformed the vocabulary of the queries to the direct statements vocabulary. We wanted to have queries with negation of various shapes. For this reason, from the BGPs of the queries we generated different sets of queries with negation, differing in the triple patterns that are negated:

– $\mathbf{Q}_{\texttt{oneTP}}$, the last triple pattern is negated;

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

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

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

The number of triple patterns negated was set to at most three, which was reasonable, since most real-world queries are of length up to three [12]. We projected out all variables in the positive part to correspond to graph pattern evaluation.

*Completeness Statements* We used two different methods of generating completeness statements depending on whether we wanted to perform either answer soundness or pattern soundness checking. As for the *generation of statements for answer soundness*, we wanted to perform it in such a way that there will be a variety of sound and possibly unsound answers. So, we generated the statements as follows: *(i)* given a query, we evaluated the query and obtained all the answer-mappings; *(ii)* for 25% of these answer-mappings, we applied them to the BGP of each NOT-EXISTS pattern of the query and constructed completeness statements out of these instantiated BGPs. This way, we can guarantee that these 25% answer-mappings are sound, while the remainder mappings are possibly unsound.

In this setting, we can naturally represent completeness statements by completeness templates (see Subsection 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 on how we get BGPs for completeness statements: instead of taking the whole instantiated BGP of the NOT-EXISTS pattern, we also generated completeness statements *separately per triple pattern* in the instantiated BGP. The first triple pattern[26] 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.

[26] We fixed an ordering.

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

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

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

### *7.2.2. Experimental Results and Discussion*

To get an idea of how soundness checking performs *without our optimization techniques,* we first ran experiments to measure the runtime of pattern soundness and answer soundness checking with no optimization of the twoTPsTO and threeTPsTO cases. Here, we set the timeout to 5 minutes. For pattern soundness checking, the median runtime for the twoTPsTO case was about 1.5 s and for the threeTPsTO case about 1.2 s. It is already reasonably fast, likely due to the fact that pattern soundness checking need not see the data graph, and depends solely on the query and com-

Table 3

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 milliseconds.

| Case | $\|\mathcal{C}\|$ | $N_Q$ | $\|Q\|$ | $\|[\![Q]\!]_G\|$ | $t_Q$ | $t_{AS}$ | $t_{AS/a}$ | $t_{PS}$ |
|---|---|---|---|---|---|---|---|---|
| oneTP | 37,769 | 57 | 3 | 24 | 14 | 1.57 | 0.069 | 0.19 |
| oneTPoneTP | 119,462 | 25 | 3 | 82 | 47 | 5.8 | 0.073 | 0.37 |
| twoTPsTO | 126,320 | 39 | 3 | 180 | 12.7 | 43.3 | 0.27 | 0.21 |
| twoTPsSE | 138,705 | 39 | 3 | 180 | 12.7 | 17.3 | 0.1 | 0.21 |
| threeTPsTO | 93,080 | 13 | 4 | 12,099 | 114 | 3,873 | 0.68 | 0.23 |

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

Now let us see the performance of soundness checking *with all our optimizations.* Table 3 summarizes the results of the experiments for all the five cases. Among those cases, the number of statements generated varies, with around 37,000 for Case oneTP, and over 93,000 for the others. Likewise, the number of queries also varies (recall that we do not include queries producing 0 answer) with Case oneTP having the most and Case threeTPsTO having the least. The median length of queries is either 3 or 4, and the median size of query results varies from around 24 to 12,099. Median query evaluation time ranges from 12 ms to 114 ms.

Median *pattern soundness* checking always takes less than a millisecond, which is more than 1000× faster than the check without optimization. The term-relevance principle probably helps rule out irrelevant completeness statements before performing the actual check.

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

Let us look more closely at answer soundness checking. Figure 2 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 ns for the time). 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 over a realistic setting based on Wikidata. We have optimized reasoning by representing sets of completeness statements using templates and by using hashmaps to apply only potentially useful statements and templates. As a result, pattern soundness checking can be done quickly, whereas answer soundness checking, though

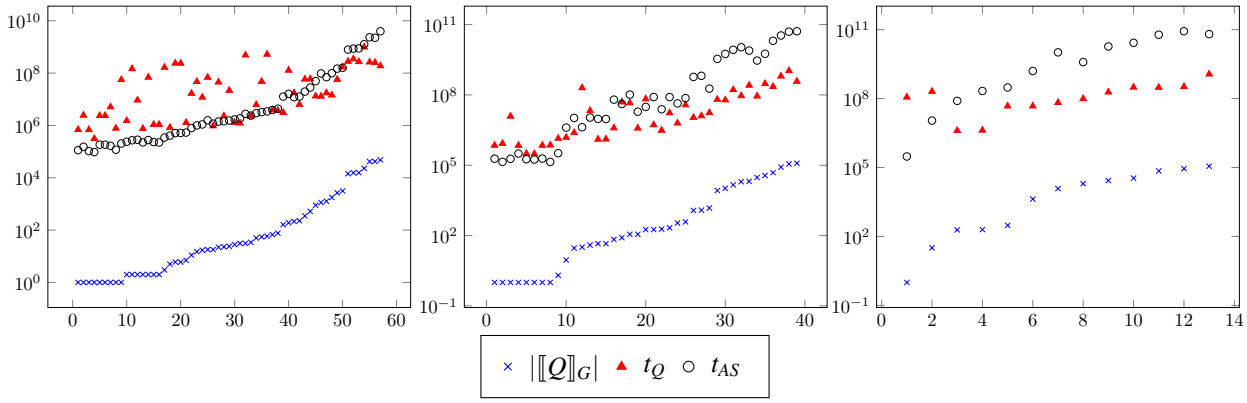



Fig. 2. Comparison between the number of query answers ($|[\![Q]\!]_G|$), query evaluation time ($t_Q$), and answer soundness checking time ($t_{AS}$) for the experiment cases: `oneTP`, `twoTPsTO`, and `threeTPsTO`. The x-axis is for query rank (from the lowest to the highest of number of query answers) and the y-axis is for number of answers or runtime in ns.

slower than pattern soundness checking, can still be done relatively fast. Moreover, the performance of answer soundness checking positively correlates with the number of query answers. Our optimization techniques have been shown to give a significant speed-up over both reasoning problems. We would also recommend that in practice, before applying answer soundness checking, pattern soundness checking should be done first since it takes less time, and by Proposition 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.

*Query Completeness* Data completeness concerns the breadth, depth, and scope of information [13] and is deemed to be one of the most significant data quality dimensions [14]. In the field of relational databases, Motro [15] and Levy [16] were among the first to investigate data completeness. Motro developed a sound technique to check query completeness based on database views, while Levy introduced the notion of local completeness statements to denote which parts of a database are complete. Razniewski and Nutt [17] further extended their results by reducing completeness reasoning to containment checking, for which many algorithms are known, and characterizing the complexity of reasoning for different classes of queries. In [18], 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.

We now move on to the Semantic Web. Fürber and Hepp [19] 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 [20]. 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. [21] 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. [22] 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 com-

pleteness (and soundness). In the context of crowdsourcing, Chu et al. [23] 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. [24] 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. [25] 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. [26] (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 [27], where a tuple is included in the complement of a relation if it is not explicitly given in the relation. Reiter [28] and Clark [29] 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, 30]. 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') [31], and the classical negation ('the negated fact truly holds') [32] 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. [33] 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. [34] 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. [35]. 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 [36], Berlin SPARQL Benchmark (BSBM) [37], and FedBench [38], 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. [39] 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.

## 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 [40]. Galárraga et al. [41] 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. [42, 43] developed techniques for relation cardinality extraction from text, which can be leveraged to generate completeness statements in the following way: when the extracted cardinality 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 [44], 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.[27] 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 [45]. 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.

*Blank Nodes* The use of blank nodes in RDF has been a controversial topic in the Semantic Web community [46, 47]. In Linked Data applications, blank nodes add complexity to data processing and data interlinking due to the local scope of their labels [48, 49]. 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 [50]. Nevertheless, blank nodes are used in practice to some degree: *(i)* for modeling unknown nulls [45, 50], and *(ii)* for modeling $n$-ary relations as auxiliary instances in reification [51].

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, 52, 53], 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 [50]). Nevertheless, in practice Semantic Web developers often tend to directly use IRIs instead of blank nodes for representing auxiliary resources, for instance by Wikidata [49].[28]

*RDFS Extension* RDFS [54] adds lightweight semantics to describe the structure and interlinking of data, usually sufficient for Linked Data publishers [48]. Main RDFS inference capabilities consist of

[27] as of Feb 18, 2017

[28] 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.

class and property hierarchies, as well as property domains and ranges [48, 55], which are widely used in practice [56]. 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 Subsection 4.1) via the closure operation wrt. RDFS ontologies [55]. 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 [55].

## 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.

## References


[1] F. Darari, S. Razniewski, R.E. Prasojo and W. Nutt, Enabling fine-grained RDF data completeness assessment, in: *ICWE*, 2016.

[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: *ISWC*, 2013. ISBN 978-3-642-41334-6.

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

[6] 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.

[7] 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.

[8] F. Darari, S. Razniewski and W. Nutt, Bridging the Semantic Gap between RDF and SPARQL using Completeness Statements, in: *ISWC Posters & Demonstrations Track*, 2014.

[9] 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.

[10] L. Rietveld and R. Hoekstra, Man vs. machine: Differences in SPARQL queries, in: *USEWOD*, 2014.
[11] M. Saleem, M.I. Ali, A. Hogan, Q. Mehmood and A.N. Ngomo, LSQ: The Linked SPARQL Queries Dataset, in: *ISWC*, 2015.
[12] 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.
[13] 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.
[14] C. Batini and M. Scannapieco, *Data and Information Quality - Dimensions, Principles and Techniques*, Data-Centric Systems and Applications, Springer, 2016.
[15] A. Motro, Integrity = Validity + Completeness, *ACM Trans. Database Syst.* **14**(4) (1989), 480–502.
[16] A.Y. Levy, Obtaining Complete Answers from Incomplete Databases, in: *VLDB*, Morgan Kaufmann, 1996, pp. 402–412. ISBN 1-55860-382-4.
[17] S. Razniewski and W. Nutt, Completeness of Queries over Incomplete Databases, *PVLDB* **4**(11) (2011), 749–760.
[18] 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.
[19] C. Fürber and M. Hepp, SWIQA - a Semantic Web Information Quality Assessment Framework, in: *ECIS 2011*, 2011.
[20] 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.
[21] P.N. Mendes, H. Mühleisen and C. Bizer, Sieve: Linked Data Quality Assessment and Fusion, in: *EDBT/ICDT Workshops*, 2012, pp. 116–123.
[22] 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.
[23] 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.
[24] 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.
[25] 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.
[26] 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.
[27] E.F. Codd, Relational Completeness of Data Base Sublanguages, *In: R. Rustin (ed.): Database Systems* (1972).
[28] 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. ISBN 978-1-4684-3384-5.
[29] K.L. Clark, Negation as Failure, in: *Logic and Data Bases*, 1978, pp. 113–141.
[30] E. Prud'hommeaux and A. Seaborne (eds), *SPARQL Query Language for RDF*, W3C Recommendation, 15 January 2008.
[31] R. Reiter, Towards a Logical Reconstruction of Relational Database Theory, in: *On Conceptual Modelling (Intervale)*, 1982, pp. 191–233.
[32] M. Gelfond and V. Lifschitz, Classical Negation in Logic Programs and Disjunctive Databases, *New Generation Comput.* **9**(3/4) (1991), 365–386.
[33] A. Polleres, C. Feier and A. Harth, Rules with Contextually Scoped Negation, in: *ESWC*, 2006.
[34] C. Lutz, I. Seylan and F. Wolter, Ontology-Based Data Access with Closed Predicates is Inherently Intractable (Sometimes), in: *IJCAI*, 2013.
[35] 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.
[36] 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.
[37] C. Bizer and A. Schultz, The Berlin SPARQL Benchmark, *Int. J. Semantic Web Inf. Syst.* **5**(2) (2009), 1–24.
[38] 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.
[39] 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.
[40] 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.
[41] 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.
[42] 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.
[43] P. Mirza, S. Razniewski, F. Darari and G. Weikum, Cardinal Virtues: Extracting Relation Cardinalities from Text, in: *ACL 2017 Short Papers*, 2017.
[44] R.E. Prasojo, F. Darari, S. Razniewski and W. Nutt, Managing and Consuming Completeness Information for Wikidata Using COOL-WD, in: *COLD*, 2016.
[45] 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.
[46] 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.

[47] Richard Cyganiak: Blank nodes considered harmful, Accessed: 2017-01-15 from http://richard.cyganiak.de/blog/2011/03/blank-nodes-considered-harmful/.
[48] 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.
[49] 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.
[50] 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.
[51] 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/.
[52] 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/.
[53] 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.
[54] 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/.
[55] S. Muñoz, J. Pérez and C. Gutierrez, Simple and Efficient Minimal RDFS, *J. Web Sem.* **7**(3) (2009), 220–234.
[56] 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.
[57] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., New York, USA, 1990. ISBN 0716710455.

## Appendix A. Proof of Proposition 1

**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 $\mu$ 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 $\mu$ 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 $\mu$ 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 $\mu$ was arbitrary, we can therefore conclude that $[\![P]\!]_{G'} \subseteq [\![P]\!]_G$. □

## Appendix B. Proof of Proposition 3

**Proposition 3** (Equivalent Partial Grounding)**.** *Let $\mathcal{C}$ be a set of completeness statements, $G$ a graph, and $(P, \nu)$ 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, $[\![(P,\emptyset)]\!]_{G'} = \bigcup_{\mu\in[\![cruc_{\mathcal{C},G}(P)]\!]_{G'}} [\![(\mu P,\mu)]\!]_{G'}$. Thus, we conclude that

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

□

## Appendix C. Proof of Lemma 1

**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:*

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

*Proof.* ($\Rightarrow$) By contrapositive. Suppose $\tilde{P}\not\subseteq G$. 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 since $\tilde{P}\not\subseteq G$, 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 $\tilde{P}\subseteq G$. It is trivial to see that $P$ is ground (i.e., has no variables), and $P\subseteq G$. Therefore, it is the case that for all extension pairs $(G,G')$, the inclusion $[\![P]\!]_{G'}\subseteq[\![P]\!]_G$ holds, implying $(G,G')\models Compl(P)$. By definition, $\mathcal{C},G\models Compl(P)$ 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 $(G,G')$. □

## Appendix D. Proof of Theorem 1

**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. $\widetilde{\mu P} \subseteq G$, *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 $\widetilde{\mu P} \not\subseteq G$. From Proposition 4, we have that $\mu P$ 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. Since $\mathcal{C}, G \not\models Compl(\mu P)$, we have the extension pair $(G, G \cup \widetilde{\mu P})$ as a counterexample for $\mathcal{C}, G \models Compl(P)$.

($\Leftarrow$) By the first claim of Proposition 4, we have that $\mu P$ 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 second claim of Proposition 4, we have that $\mathcal{C}, G \models Compl(P)$. □

## Appendix E. Proof of Proposition 5

**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* $\Pi_2^p$*-complete.*

*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 $\mu$ 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 CoNP that $(G, G \cup \mu P) \models \mathcal{C}$. If it holds, then $\mathcal{C}, G \not\models Compl(P)$ by the counterexample $G' = G \cup \mu P$.

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 $\gamma_i$ 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 $\psi$: Unfold the universally quantified variables $x_1, \ldots, x_m$ in $\psi$, and then check if for every formula in the set $\Psi_{unfold}$ 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[29]

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

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 $\psi$.

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 $\psi$. For each propositional variable $p_i$, we encode the positive literal $p_i$ as the variable $var(p_i) = ?p_i$

[29] 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`.

and the negative literal $\neg p_i$ as the variable $var(\neg p_i) = ?\neg p_i$. Given a clause $\gamma_i = l_{i1} \vee l_{i2} \vee l_{i3}$, the operator $tp(\gamma_i)$ maps $\gamma_i$ to $(var(l_{i1}), var(l_{i2}), var(l_{i3}))$. We then define the following BGP to encode the structure of $\psi$:

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

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

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

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

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

We are now ready to construct the following completeness statement:

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

In summary, our encoding consists of the following ingredients: the set $\mathcal{C} = \{\, C_\forall, C_\psi \,\}$ 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 $\psi$. 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 $\psi$. Each of the partially mapped BGPs corresponds to an assignment for the universally quantified variables in the set $\Psi_{unfold}$ of the unfolding results of $\psi$.

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 $\nu$ from the encoding $?y_i$ of the existentially quantified variables in $P_\psi$ such that $\nu(\mu P_\psi) \subseteq P_{true}$. Note that the mapping $\nu$ simulates a satisfying assignment for the corresponding existentially quantified formula in the set $\Psi_{unfold}$. 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 $\Psi_{unfold}$, the formula $\psi$ 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 $\nu$ from the encoding $?y_i$ of the existentially quantified variables in $P_\psi$ 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 $\Psi_{unfold}$. This implies that $\psi$ 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 $\mathcal{C},G \not\models Compl(P)$.

□

## Appendix F. Proof of Proposition 6

**Proposition 6.** *Deciding the entailment $\mathcal{C},G \models Compl(P)$, given a set $\mathcal{C}$ of completeness statements, a fixed graph $G$, and a BGP $P$, is $\Pi_2^P$-complete.*

*Proof.* The membership follows immediately from Proposition 5, while the hardness follows from the reduction proof of that proposition, in which the graph is fixed. □

## Appendix G. Proof of Proposition 7

**Proposition 7.** *Deciding the entailment $\mathcal{C},G \models Compl(P)$, given a set $\mathcal{C}$ of completeness statements, a graph $G$, and a fixed BGP $P$, is NP-complete. The complexity remains the same even 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 there-

fore 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 directed graphs, which is known to be NP-hard [57]. We encode the problem graph $G_p = (V, E)$, i.e., the directed graph we want to check whether it is 3-colorable, as a set $\mathit{triples}(G_p)$ of triple patterns. We associate to each vertex $v \in V$, a new variable $?v$. Then, we define $\mathit{triples}(G_p)$ as the union of all triple patterns $(?s, \mathit{edge}, ?o)$ created from each pair $(s, o) \in E$ where $?s$ is the associated variable of $s$, *edge* is an IRI and $?o$ is the associated variable of $o$. Let the BGP $P_{col}$ be:

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

Next, we create the completeness statement

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

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

The problem graph $G_p$ is 3-colorable iff

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

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

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

## Appendix H. Proof of Proposition 8

**Proposition 8.** *Deciding the entailment $\mathcal{C}, G \models \mathit{Compl}(P)$, given a fixed set $\mathcal{C}$ of completeness statements, a graph $G$, and a BGP $P$, is CoNP-complete. The complexity stays the same even when the graph is fixed.*

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

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

We guess a mapping $\mu$ over $P$ such that $\mu P \not\subseteq G$, which implies that $(G, G \cup \mu P) \not\models \mathit{Compl}(P)$. 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 $\mathcal{C}, G \not\models \mathit{Compl}(P)$ 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 directed graphs. We encode the problem graph $G_p = (V, E)$, i.e., the directed graph for which we want to check whether it is 3-uncolorable, as a set $\mathit{triples}(G_p)$ of triple patterns. We associate to each vertex $v \in V$, a new variable $?v$. Then, we define $\mathit{triples}(G_p)$ as the union of all triple patterns $(?s, \mathit{edge}, ?o)$ created from each pair $(s, o) \in E$ where $?s$ is the associated variable of $s$, *edge* is an IRI and $?o$ is the associated variable of $o$. Let the BGP $P$ be:

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

Let the graph $G$ be the color graph:

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

Next, we create the completeness statement

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

Then, the following claim holds:

The problem graph $G_p$ is 3-uncolorable iff

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

*Proof of the claim*: ($\Rightarrow$) The proof relies on Algorithm 1 and Theorem 1. Assume $G_p$ is 3-incolorable. By construction, the part $\mathit{triples}(G_p)$ of the BGP $P$ can be grounded completely due to the statement $C$, that is, the crucial part operator *cruc* returns exactly that part.

However, as $G_p$ is 3-uncolorable, there is no mapping $\mu$ 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. Assume that $G_p$ is 3-colorable. Thus, there must be a mapping $\mu$ 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 $\mu$ arbitrarily. By construction, the part $triples(G_p)$ of the BGP $P$ can be grounded completely due to the statement $C$, that is, the crucial part operator *cruc* returns exactly that part. Since the graph $G_p$ is 3-colorable, we can then reuse the mapping $\mu$ for mapping $triples(G_p)$ to $G$. The *epg* operator results therefore include that mapping, which is then applied to the remaining part of $P$, that is, the triple pattern $(a, b, c)$. Note that the triple pattern consists only of constants, so the mapping application has no effect. Now we have to check the completeness of $(a, b, c)$. As no completeness statements can be evaluated over that remaining part, it is then the case that we are already saturated for $(a, b, c)$. By Theorem 1, the BGP $P$ can be guaranteed to be complete iff all saturated instantiations wrt. $\{C\}$ are in $G$. However, clearly $(a, b, c)$ is not in $G$. Thus, we have that $\{C\}, G \not\models Compl(P)$. □

## Appendix I. Proof of Proposition 9

**Proposition 9.** *Deciding the entailment $\mathcal{C}, G \models Compl(P)$, given a fixed set $\mathcal{C}$ of completeness statements, a graph G, and a fixed BGP P, 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 J. Proof of Theorem 2

**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 $\nu$ 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, $(G, G \cup \nu\mu P_w) \not\models Sound(\mu, P)$. □

## Appendix K. Proof of Proposition 10

**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 $\mu$ 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 Subsection 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 $\mathcal{C} \models Compl(P \mid P')$. □

## Appendix L. Proof of Lemma 2

**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 $\mathcal{C} \models Compl(P_i \mid P^+)$. 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 $\mathcal{C} \models Compl(P_i \mid P^+)$ and $(G, G') \models \mathcal{C}$, it is the case that $(G, G') \models Compl(P_i \mid P^+)$. Consequently, by $[\![(var(P_i), P_i \cup P^+)]\!]_{G'} \subseteq_s [\![P_i]\!]_G$ 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 $\mu \in [\![P]\!]_{G'}$. □

## Appendix M. Proof of Theorem 3

**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* $P_i \in P^-$.

*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 $\mathcal{C} \not\models Compl(P_w \mid P^+)$. 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 $(G, G') \not\models Sound(P)$.

By the definition of $T_{\mathcal{C}}$, it holds that $(G, G') \models \mathcal{C}$. We now have to show that $(G, G') \not\models Sound(P)$. 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 $[\![\tilde{id} P_i]\!]_{\tilde{P}^+ \cup T_{\mathcal{C}}(\tilde{P}_w \cup \tilde{P}^+)} = \emptyset$. Due to the consistency of $P$ and the non-containment property between different negation parts, due to the absence of redundant negations in an NRF graph pattern, there is no negation part $P_j \neq P_w$ 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

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

However, this holds from the consistency of $P$ and 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 $(G, G') \models Sound(P)$. □