Certain Answers for SPARQL with Blank Nodes

Daniel Hernández, Claudio Gutierrez, and Aidan Hogan(✉)

IMFD Chile and Department of Computer Science, University of Chile, Santiago, Chile
daniel@degu.cl, aidhog@gmail.com

Abstract. Blank nodes in RDF graphs can be used to represent values known to exist but whose identity remains unknown. A prominent example of such usage can be found in the Wikidata dataset where, e.g., the author of Beowulf is given as a blank node. However, while SPARQL considers blank nodes in a query as existentials, it treats blank nodes in RDF data more like constants. Running SPARQL queries over datasets with unknown values may thus lead to counter-intuitive results, which may make the standard SPARQL semantics unsuitable for datasets with existential blank nodes. We thus explore the feasibility of an alternative SPARQL semantics based on certain answers. In order to estimate the performance costs that would be associated with such a change in semantics for current implementations, we adapt and evaluate approximation techniques proposed in a relational database setting for a core fragment of SPARQL. To further understand the impact that such a change in semantics may have on query solutions, we analyse how this new semantics would affect the results of user queries over Wikidata.

1 Introduction

Incomplete information poses a major challenge for data management on the Web. Web data may be incomplete for a variety of reasons: the missing information may be unknown to those who created the dataset, it may be suppressed for privacy reasons, it may not yet have been added to the dataset, it may be a gap left after integrating other datasets, and so forth. A fundamental question for exploiting data on the Web is then how to define the semantics for (and process) queries over incomplete datasets. An important notion of incompleteness is that of *unknown values*. To take a literary example, we know that the poem "Beowulf" was written by *somebody*, but nobody knows who. One option is to simply omit authorship, but we would then lose valuable information that "Beowulf" has *some* author. Various works have then been proposed to deal with incomplete information [10,11,20], amongst which are recent works proposing query rewritings to provide sound answers over databases with unknown values [8,11,12,19]; such works have focused on relational settings.

On the other hand, there is a strong need for methods to deal with incomplete information and unknown values in a Semantic Web setting. In RDF, blank nodes

© Springer Nature Switzerland AG 2018
D. Vrandečić et al. (Eds.): ISWC 2018, LNCS 11136, pp. 337–353, 2018.
https://doi.org/10.1007/978-3-030-00671-6_20

can be used either to represent a resource for which no IRI is defined, or as an existential to represent an unknown value [15]. The RDF standard specifically defines blank nodes with an existential semantics [14]. However, SPARQL [13] does not follow the standard existential semantics of blank nodes in RDF data. As a result, when SPARQL queries are run over datasets where blank nodes are used as existentials to represent unknown values, the results can be unintuitive (or arguably *incorrect* [11]). We now provide such an example taken from the Wikidata [22] knowledge-base, which publishes data associated with Wikipedia as RDF and provides a public SPARQL query interface on the Web.

Example 1. Take the following RDF triples from Wikidata [22][1] and the following two SPARQL queries, which we will denote Q_1 (above) and Q_2 (below):

```
w:NicoleSimpson  w:killedBy _:b .
w:NicoleSimpson  w:gender   w:Female .
w:ReevaSteenkamp w:killedBy w:OscarPistorius .
w:ReevaSteenkamp w:gender   w:Female .
w:OJSimpson      w:gender   w:Male .
w:OscarPistorius w:gender   w:Male .
```

```
SELECT ?victim WHERE
{ ?victim w:killedBy ?person .
  ?person w:gender w:Male . }
```

```
SELECT ?victim WHERE
{ ?victim w:killedBy ?person .
  FILTER NOT EXISTS
  { ?person w:gender w:Male . } }
```

In the data, the blank node (`_:b`) denotes that Nicole Brown Simpson (a victim of homicide) has a killer, but that her killer is unknown. For Q_1, SPARQL dictates a single solution – {`?victim/w:ReevaSteenkamp`} – which can be considered a *certain answer*; here the (unknown) killer of Nicole Simpson *may* have been male, but *uncertain answers* of this form are not returned. On the other hand, for Q_2, SPARQL again dictates a single solution – {`?victim/w:NicoleSimpson`} – but this answer is *uncertain* by the same reasoning: we do not know that the killer of Nicole Simpson was not male; `_:b` could refer to a male in the data. □

These two examples highlight a key problem in the current SPARQL semantics when dealing with unknown values. In the first query only certain answers are returned: answers that hold no matter to whom the unknown value(s) refer(s). In the second query uncertain answers are returned: answers that may or may not hold depending on whom the unknown value(s) refer(s) to. The key premise of this paper is to then ask: should users be offered a *choice* to only return certain answers? Is such a choice important? And what would be its cost?

Regarding cost, unfortunately, query evaluation under certain answer semantics incurs a significant computational overhead; for example, considering queries expressed in the standard relational algebra, if we consider only "complete databases" without unknown values, the data complexity of the standard query evaluation problem is AC^0; on the other hand, the analogous complexity with unknown values under certain answer semantics leaps to coNP-hardness [1].

One can thus hardly blame the design committee of the SPARQL language for choosing to initially overlook the issue of unknown values: not only would the complexity of query evaluation have escalated considerably under something

[1] While the example uses real data, for readability, we use fictitious IRIs. In reality, Wikidata uses internal identifiers, such as `w:Q268018` to represent Nicole Simpson.

well-founded like a certain-answer semantics, the cost and complexity of correctly implementing the new standard would likewise have jumped considerably. Still, in this paper, we propose that it is time to revisit the issue of evaluating SPARQL queries in the presence of unknown values (blank nodes) in the data.

In terms of need, a recent study of blank nodes suggests that 66% of websites publishing RDF use blank nodes, with the most common use-cases being to represent resources for which no IRI has been defined (e.g., for representing RDF lists), or to represent unknown values [15]. The methods proposed in this paper specifically target datasets using blank nodes in the second sense; such datasets include Wikidata [22], as illustrated in Example 1.

In terms of cost, work by Guagliardo and Libkin [11] offers promising results in terms of the practical feasibility of *approximating* certain answers in the context of relational databases, returning only (but not all) such answers. Performance results suggest that such approximations have reasonable runtimes when compared with standard SQL evaluation. Furthermore, their implementation strategy is based on query rewriting over off-the-shelf query engines, obviating the need to build special-purpose engines, minimising implementation costs.

In this paper, we thus tackle the question: should users be given a choice of certain semantics for SPARQL? Along these lines, we adapt the methods of Guagliardo and Libkin [11] in order to propose and evaluate the first approach (to the best of our knowledge) that guarantees to return only certain answers for a fragment of SPARQL (capturing precisely the relational algebra) over RDF datasets with existential blank nodes, further developing a set of concrete rewriting strategies for the SPARQL setting. We evaluate our rewriting strategies for two popular SPARQL engines – Virtuoso and Fuseki – offering comparison of performance between base queries and rewritten queries (under various strategies), and a comparison of our SPARQL and previous SQL results [11]. We further conduct an analysis of Wikidata user queries to see if a certain answer semantics would really affect the answers over real-world queries and data, performing further experiments to ascertain costs in this setting.

2 Related Work

The conceptual problem of evaluating queries over data with unknown values is well-known in the relational database literature from as far back as the 70's, where Codd presented an extension of the relational model to allow nulls to encode unknown values [7]. Work on querying data with unknown values has continued throughout the decades, mostly in the context of relational databases (e.g., [17,19]), but also in various semi-structured settings (e.g., [2,9]).

A recent milestone has been the development of methods for *approximating* certain answers, where the current work is inspired by the proposal of Guagliardo and Libkin [11] of a method to (under-)approximate certain answers for SQL. Their goal is to trade completeness of certain answers against efficiency, ensuring that only (but not necessarily all) certain answers are returned in the presence of unknown values. We thus adapt their techniques for the SPARQL setting over RDF graphs with unknown values and propose SPARQL-specific rewriting strategies. In the experimental section, we provide a high-level comparison of our results for SPARQL with those published by Guagliardo and Libkin for SQL.

We are not the first work to explore a certain answer semantics for SPARQL. Ahmetaj et al. [2] define a certain answer semantics for SPARQL, but their focus is on supporting OWL 2 QL entailment for queries based on *well-designed patterns* [21], and in particular on complexity results for query evaluation, containment and equivalence. Arenas and Perez [5] also consider a certain answer semantics for SPARQL towards studying conditions for *monotonicity*: a semantic condition whereby answers will remain valid as further data is added to the system; as such, certain answers in their work are concerned with an *open world semantics*. However, determining if a query is *weakly monotonic* – i.e., monotonic disregarding unbound values – is undecidable. Hence Arenas and Ugarte [6] later propose a syntactic fragment of SPARQL that closely captures this notion of weak monotonicity. In contrast to such works, we maintain SPARQL's negation features [3] with a closed world semantics. In many contexts, users are interested in writing SPARQL queries with respect to what the present dataset does/does not contain; where, e.g., as we discuss later, many of the use-case queries for the Wikidata SPARQL service use non-monotonic features, such as difference.

To our knowledge, this is the first work to investigate a certain answer semantics for SPARQL considering existential blank nodes in RDF data.

3 Preliminaries

RDF and Incomplete Information: We assume three pairwise disjoint sets: $\mathbf{B}$ of blank nodes, $\mathbf{C}$ of constants, and $\mathbf{V}$ of attribute names (considered to represent variables when we later speak of queries). We will henceforth refer to blank nodes as simply "blanks". We denote blanks as $\bot_1, \bot_2, \bot_3$, etc.; constants with lowercase a, b, c, etc.; and attribute names with uppercase X, Y, Z, etc.

We define a tuple to be a function (a mapping) $\mu : \mathbf{V} \to (\mathbf{C} \cup \mathbf{B})$; for simplicity, we will denote the mapping μ with domain $\{X, Y\}$ where $\mu(X) = a$ and $\mu(Y) = b$ simply as $XY \mapsto (a, b)$, or even as the tuple (a, b) if the attributes X and Y are clear from the context. A relation R is determined by a set of tuples with the same domain (we need not consider empty relations).

An RDF graph G (or simply a graph) then corresponds to an instance of a single ternary relation with fixed attributes S, P, O, called *subject*, *predicate* and *object*, where P can only map to $\mathbf{C}$ (i.e. no blanks can occur as predicate), while S and O can map to values from $\mathbf{C} \cup \mathbf{B}$. Here $\mathbf{C}$ represents both IRIs and RDF literals; such a distinction of RDF constants is not exigent for us.

In this model, features for encoding incomplete information – specifically unknown values – are introduced by the semantics of blanks. Here the set $\mathbf{B}$ of blanks appearing in the data are interpreted as existentially-quantified variables in a manner consistent with the RDF semantics [14]; this is how, e.g., Wikidata uses blanks to represent unknown values. Blanks are synonymous with *marked nulls* in a relational setting: nulls that can appear in various locations.

Hence RDF graphs with blanks correspond to (ternary) relations with marked nulls, which have been called *naive tables*, *v-tables*, or *e-tables* by various authors (see [16]); here we will refer to them as *v-tables*. We say a v-table/graph is *complete/ground* when no nulls/blanks are used. With this correspondence established, we may henceforth use terms such as graph/table or blank/null interchangeably as best fits the particular context.

The semantics of a v-table is then defined as follows. A valuation is a mapping $v : \mathbf{C} \cup \mathbf{B} \rightarrow \mathbf{C}$ such that $v(c) = c$ for every $c \in \mathbf{C}$. Valuations are extended to tuples, relations and databases in the natural way. For instance, applying a valuation v to a graph G, denoted $v(G)$, results in the complete graph derived by replacing every blank $\perp_i$ in G by $v(\perp_i)$. The semantics of a v-table $[\![R]\!]$ is then given as the ground relations $\{v(R) : v \text{ is a valuation}\}$.

A Core SPARQL Algebra: We now define an algebra for the fragment of SPARQL considered, focusing on set semantics. We first define queries for ground graphs where unknown values will be treated later: A query is a combination of the following algebraic operations: selection (σ_θ), renaming ($\rho_{X/Y}$), projection ($\pi_{\bar{X}}$), natural join ($\bowtie$), union ($\cup$) and difference ($\setminus$). Attribute names ($\mathbf{V}$) are then synonymous with variables in SPARQL. The condition θ of a selection $\sigma_\theta(R)$ is a Boolean combination ($\wedge$,$\vee$,$\neg$) of terms of the form $X = Y$ or $X = c$, where X and Y refer to the attributes of R and $c \in \mathbf{C}$. We refer to this fragment as “SRPJUD” capturing the initials of the operations allowed; this fragment corresponds directly to the relational algebra but will be applied to graphs in a SPARQL setting. The correspondence between SRPJUD operators and syntactic SPARQL features is shown in Table 1. Note however that union and difference in SRPJUD follow the relational algebra in that – unlike standard SPARQL – $R \cup S$ and $R \setminus S$ assume that the relations R and S have the same attributes. Thus, SRPJUD does not support generating unbound variables through `UNION` as supported by SPARQL for attributes not in both R and S. However the difference operator in SRPJUD and the *outer-difference* operator in SPARQL can be mutually expressed using other SRPJUD operators. Taking difference, $R \setminus S$ is a particular case of the outer difference $R - S$ when the attributes of R and S are the same. Conversely, letting $\bar{X}$ denote the set of common attributes of R and S, the outer-difference $R - S$ can be expressed as (1) $R \bowtie (\pi_{\bar{X}}(R) \setminus \pi_{\bar{X}}(S))$ when $\bar{X}$ is non-empty or (2) R when $\bar{X}$ is empty.

Finally, given a query Q expressed in SRPJUD and a graph G, we write $Q(G)$ to denote the result of evaluating Q over the graph G following standard conventions for the relational algebra. Our focus will then be on answering queries in the core SRPJUD fragment over graphs with unknown values. We leave support for the following features of SPARQL for future work: (1) Bag semantics. (2) SPARQL unbounds as created by either SPARQL `UNION` or `OPT`. (3) Other features such as property paths, solution modifiers, aggregations, other filter expressions, named graphs, etc. These latter SPARQL features (e.g., aggregation) can, however, be defined *syntactically* on top of the core SRPJUD algebra.

Table 1. Mapping between SPARQL and SRPJUD

$\{ X \ p \ Y \} \leftrightarrow \rho_{S/X}(\rho_{O/Y}(\pi_{S,O}(\sigma_{P=p}(G))))$	
P `.` $Q \leftrightarrow P \bowtie Q$	`SELECT` $\bar{X}$ `WHERE` $P \leftrightarrow \pi_{\bar{X}}(P)$
P `MINUS` $Q \leftrightarrow P - Q$	`SELECT (`X `AS` Y`) WHERE` $P \leftrightarrow \rho_{X/Y}(P)$
P `UNION` $Q \leftrightarrow P \cup Q$	P `FILTER` $\theta \leftrightarrow \sigma_\theta(P)$

Certain and Possible Answers: We now define a certain- and possible-answer semantics for SPARQL where unknown values are present in the RDF data in the form of blanks. Let Q be a query and G be a graph with blanks. Then a widely used definition of *certain answers* are the answers μ of $Q(G)$ such that $\mu \in Q(v(G))$ for every valuation v. Another more general definition of certain answers – first defined by Lipski [20] and called *certain answers with nulls* by Libkin [18] – states that μ is a certain answer of $Q(G)$ iff $v(\mu) \in Q(v(G))$ for every valuation v; this semantics allows for returning unknown values in answers.

Example 2. Consider an RDF graph G with $\{(a, b, c), (a, d, \bot_1)\}$ and a query $\pi_{\{P,O\}}(G)$. Under *certain answer semantics*, $\{(b, c)\}$ is returned. Under *certain answer semantics with nulls*, $\{(b, c), (d, \bot_1)\}$ is returned; here $\bot_1$ is interpreted as stating that for all valuations, there exists *some* constant there. □

A complementary notion is that of a *possible answer*: a tuple μ is a possible answer of $Q(G)$ if there exists a valuation v such that $v(\mu) \in Q(v(G))$.

Example 3. Consider again the graph G from Example 2 but instead consider a query $\pi_{\{S,O\}}(\sigma_{P=b}(G)) \setminus \pi_{\{S,O\}}(\sigma_{P=d}(G))$. Under both certain answer semantics an empty result will be returned since there is a valuation μ such that $\mu(\bot_1) = c$. Here (a, c) will be considered a *possible* rather than a certain answer. □

Given a query Q and an RDF graph G, then we write $\text{cert}(Q, G)$ and $\text{poss}(Q, D)$ to denote respectively the sets of certain and possible answers, defined as follows:

$$\text{cert}(Q, G) = \bigcap \{\mu \mid v(\mu) \in Q(v(D)) \text{ for all valuations } v\},$$
$$\text{poss}(Q, G) = \bigcup \{\mu \mid v(\mu) \in Q(v(G)) \text{ for all valuations } v\}.$$

Note that the former definition captures certain answers *with nulls*, which we use here; also, note that $\text{cert}(Q, G) \subseteq \text{poss}(Q, G)$: certain answers are also possible.

4 Approximating Certain Answers

The problem of query evaluation under certain answers is coNP-hard (data complexity); without unknown values the analogous problem is in AC^0. Likewise the definition of the semantics does not directly suggest a practical query answering procedure. Hence in this section we explore an algebra that allows for approximating certain/possible answers based on the notion of maybe tables.

4.1 Unification

We first define the notion of *unification*, which joins tuples with unknown values. We say that μ_1 and μ_2 *unify*, denoted $\mu_1 \Uparrow \mu_2$, iff for every common attribute X that they share, it holds that $\mu_1(X) = \mu_2(X)$ or $\mu_1(X) \in \mathbf{B}$ or $\mu_2(X) \in \mathbf{B}$; in other words, $\mu_1 \Uparrow \mu_2$ holds iff there is a valuation v such that $v(\mu_1(X)) = v(\mu_2(X))$ for every common attribute X. The unification of two tuples $(\mu_1 {}^\frown \mu_2)(X)$ is defined as $\mu_1(X)$ if $\mu_2(X)$ is $\bot$, or $\mu_2(X)$ otherwise. Unification allows to extend the standard operators join, semijoin and anti-semijoin to include the semantics of nulls by replacing the concept of joinable tuples and joins of tuples by the notion of unifiable tuples and unifications of tuples:

$$\begin{aligned}
P \bowtie_{\Uparrow} Q &= \{\mu_1 {}^\frown \mu_2 \mid \mu_1 \in P, \mu_2 \in Q, \text{ and } \mu_1 \Uparrow \mu_2\},\\
P \ltimes_{\Uparrow} Q &= \{\mu_1 \in P \mid \exists \mu_2 \in Q : \mu_1 \Uparrow \mu_2\},\\
P \mathbin{\overline{\ltimes}}_{\Uparrow} Q &= \{\mu_1 \in P \mid \nexists \mu_2 \in Q : \mu_1 \Uparrow \mu_2\}.
\end{aligned}$$

Such operators will be essential to defining an approximation of certain answers, but they do not appear in SRPJUD and cannot be expressed in this fragment since it does not contain any means to distinguish blanks from constants. Hence to rewrite a SRPJUD query, we need a built-in predicate of the form $\mathrm{bk}(X)$ in the target algebra, which evaluates to true for a tuple μ if $\mu(X) \in \mathbf{B}$, or false otherwise. We can now represent $P \ltimes_{\Uparrow} Q$ as $\pi_{\bar{X}}(\sigma_{\theta_{\Uparrow}}(P \bowtie \rho_{\bar{X}/\bar{X}'}(Q)))$ and $P \mathbin{\overline{\ltimes}}_{\Uparrow} Q$ as $P - (P \ltimes_{\Uparrow} Q)$, where $\bar{X}$ denotes the attributes/variables of P, $\bar{X}'$ denotes fresh variables, and $\theta_{\Uparrow}$ will be rewritten to $(X = Y \vee \mathrm{bk}(X) \vee \mathrm{bk}(X'))$.

Translating $P \bowtie_{\Uparrow} Q$ to SRPJUD is more difficult. One option is to use the *active domain*: the set of all possible values to which blanks can be evaluated [17]; however, this would cause obvious practical problems. Hence we will rather use two non-SRPJUD features of SPARQL to implement unification: the ternary conditional operator, which allows for returning one of two values based on a condition (denoted `IF`$(\cdot,\cdot,\cdot)$ in SPARQL); and the bind operator, which can bind a new value to the relation (denoted `BIND`$(\cdot,\cdot)$ in SPARQL). From these, we derive a new operator $\mathrm{if}_{\theta,X,Y,Z}(\mu)$, which returns $\mu \cup \{Z \mapsto (\mu(X))\}$ if $\mu \models \theta$, or $\mu \cup \{Z \mapsto (\mu(Y))\}$ otherwise. The operator thus creates a new attribute Z and assigns it the value of X if the condition θ is true, otherwise it assigns it the value of Y. We call SRPJUD extended with these unification operators SRPJUD$_{\Uparrow}$. Returning to $P \bowtie_{\Uparrow} Q$, we can first apply a Cartesian product and then unify the results with the ternary conditional operator. More formally, assume that P and Q contain one shared attribute X. We can now express $P \bowtie_{\Uparrow} Q$ as $\rho_{X''/X}(\pi_{X'',\bar{Y}}(\mathrm{if}_{\mathrm{bk}(X'),X,X',X''}(\sigma_{\theta_{\Uparrow}}(P \bowtie \rho_{X/X'}(Q)))))$, where $\bar{Y}$ denotes the non-shared attributes of P and Q and $\theta_{\Uparrow}$ is as before. In this case, $P \bowtie \rho_{X/X'}(Q)$ denotes a Cartesian product since there are no shared attributes. This process extends naturally to performing unifications over multiple attributes[2].

[2] Note that given two blank nodes on either side, this approach chooses the left blank node arbitrarily and drops the other. This may lead to losing certain answers, which we accept as part of the under-approximation.

4.2 Approximations

We wish to under-approximate certain answers to guarantee that all answers returned are certain while maximising the certain answers returned. But if we consider under-approximating results to a query $P - Q$, intuitively for P we must under-approximate certain answers, while for Q we should *over*-approximate *possible* answers to Q to ensure we remove everything from P that might match under *some* valuation in Q. Note that Q might itself be a query of the form $R - S$; etc. Hence, to under-approximate certain answers for SRPJUD, we need a way to over-approximate possible answers [11]: given a query Q in SRPJUD, we will rewrite it to a pair of queries $(Q^+, Q^?)$ in SRPJUD$_\Uparrow$, under-approximating certain answers and over-approximating possible answers for Q, respectively.

The first operator we define is the selection operator, where we must take care of inequalities involving blanks; we adopt the rewriting proposed by Guagliardo and Libkin [11], and shown by them to have good performance.

Definition 1. *We define the translation of a SRPJUD query Q to a pair of* approximation queries $(Q^+, Q^?)$ *in SRPJUD$_\Uparrow$ recursively as follows:*

$$\begin{aligned} G^+ &= G, & G^? &= G,\\ (P \cup Q)^+ &= P^+ \cup Q^+, & (P \cup Q)^? &= P^? \cup Q^?,\\ (P \bowtie Q)^+ &= P^+ \bowtie Q^+, & (P \bowtie Q)^? &= P^? \bowtie_\Uparrow Q^?,\\ (P - Q)^+ &= P^+ \mathbin{\overline{\ltimes}_\Uparrow} Q^?, & (P - Q)^? &= P^? - Q^+,\\ (\sigma_\theta(P))^+ &= \sigma_{\theta^*}(P^+), & (\sigma_\theta(P))^? &= \sigma_{\neg(\neg\theta)^*}(P^?),\\ (\pi_{\bar{X}}(P))^+ &= \pi_{\bar{X}}(P^+), & (\pi_{\bar{X}}(P))^? &= \pi_{\bar{X}}(P^?),\\ (\rho_{X/Y}(P))^+ &= \rho_{X/Y}(P^+), & (\rho_{X/Y}(P))^? &= \rho_{X/Y}(P^?), \end{aligned}$$

where θ^* denotes the translation defined inductively as follows, noting that X and Y are attributes and a is some constant:

$$\begin{aligned} (X = Y)^* &= (X = Y), & (X \neq Y)^* &= (X \neq Y) \wedge \neg\,\mathrm{bk}(X) \wedge \neg\,\mathrm{bk}(Y),\\ (X = a)^* &= (X = a), & (X \neq a)^* &= (X \neq a) \wedge \neg\,\mathrm{bk}(X),\\ (\theta_1 \vee \theta_2)^* &= \theta_1^* \vee \theta_2^*, & (\theta_1 \wedge \theta_2)^* &= \theta_1^* \wedge \theta_2^*. \end{aligned}$$

□

4.3 Relation to Certain/Possible Answers

To state formally the relation of certain/possible answers with the corresponding approximation queries given in Definition 1, we require a notion of a subset of answers under unification. Importantly, the following definition is used to ensure that any tuple that unifies with a possible answer (e.g., $(\bot_1,\bot_1)$) will unify with an answer in the over-approximation (e.g., $(\bot_1,\bot_2)$).

Definition 2. *Given P and Q, we state that $P \subseteq_\Uparrow Q$ iff for each tuple $\mu \in P$, there exists $\mu' \in Q$ such that $\nu(\mu') = \mu$ for some valuation ν.* □

Lemma 1. *Let Q be a SRPJUD query and let $(Q^+, Q^?)$ be the approximation queries for Q as defined in Definition 1. Then, for any RDF graph G, it holds that $Q^+(G) \subseteq \mathrm{cert}(Q, G)$ and $Q^?(G) \supseteq_{\Uparrow} \mathrm{poss}(Q, G)$.*

Proof. Follows from induction on the structure of the query, following similar techniques as used for Lemmas 1 and 2 of [11]. □

Computing exact certain/possible answers has a high complexity, where Definition 1 directly leads to a rewriting strategy for approximating certain/possible answers. For example, to under-approximate the certain-answers of a SRPJUD query Q, we can rewrite it to the SRPJUD$_{\Uparrow}$ $Q^?$ and execute that query; furthermore, evaluating queries in SRPJUD$_{\Uparrow}$ remains tractable in data complexity per the class of base queries SRPJUD (and unlike computing exact certain answers).

5 SPARQL Rewriting Strategies

We now explore alternatives in SPARQL to express the rewriting of Definition 1. All such alternatives are equivalent; in practice however, these strategies can exhibit major performance variations when applied over SPARQL query engines.

The base case in the SPARQL translation is G, which refers to a ternary relation with fixed attributes S, P and O. The basic unit of querying in SPARQL is a *triple pattern*, e.g., XpY ($X \in \mathbf{V}$, $Y \in \mathbf{V}$, $p \in \mathbf{C}$). In RDF, the P attribute cannot take blanks, and hence we do not need to consider unification on that attribute directly. A *basic graph pattern* Q in SPARQL is a join over triple patterns $T_1 \bowtie \cdots \bowtie T_k$ where each T_i $(1 \leq i \leq k)$ is a triple pattern.

The most complex case to consider is the difference operator $\mathrm{P} - \mathrm{Q}$, where certain answers are under-approximated by the unification anti-semijoin $P^+ \,\overline{\ltimes}_{\Uparrow}\, Q^?$ (where $Q^?$ is itself over-approximated). The direct application of the translation rules produces complex queries that can be rewritten to a "friendlier" form for SPARQL engines, as now described. First, given a difference $P - Q$ we say that X is a *correlated* attribute of the difference if X is shared by P and Q. In the following we will assume that $\mathrm{P} - \mathrm{Q}$ is a difference with at least a correlated variable and that Q is a basic graph pattern.

CNF/DNF Rewritings: In the difference $P - Q$, let $Q = T_1 \bowtie T_2$ (a common case). The base translation evaluating the required unification in Q is then given as $(T_1 \bowtie T_2)^? = \beta_{\bar{X},\bar{X_1},\bar{X_2}}(\sigma_{\Theta_{\Uparrow}}(U_1 \bowtie U_2))$ where U_1 and U_2 are the respective results of replacing shared variables $(\bar{X})$ in $T_1 \bowtie T_2$ by fresh variables (denoted $\bar{X_1}$ and $\bar{X_2}$), where $\Theta_{\Uparrow}$ is a conjunction of the standard unifiable condition applied to each pair of renamed variables X_1 and X_2 for X (i.e., $\bigwedge_{X \in \bar{X}}(X_1 = X_2 \vee \mathrm{bk}(X_1) \vee \mathrm{bk}(X_2))$), and where, the operator $\beta_{\bar{X},\bar{X_1},\bar{X_2}}$ extends the solution for each $X \in \bar{X}$ using the function $\mathrm{if}_{\mathrm{bk}(X_2),X_1,X_2,X}(\cdot)$. These definitions then extend naturally (but verbosely) to the case where Q is $T_1 \bowtie_{\Uparrow} \ldots \bowtie_{\Uparrow} T_k$. This implies taking the Cartesian product of all triple patterns, filtering by a conjunction of unification conditions $\sigma_{\theta_{\Uparrow}}$, and then selecting constants over blanks.

The aforementioned unification condition $\Theta_{\Uparrow}$ is in conjunctive normal form (CNF): $\theta_1 \wedge \cdots \wedge \theta_n$ where for $1 \leq i \leq n$, each term θ_i is a disjunctive clause. An alternative solution is to rewrite the unification condition to its equivalent disjunctive normal form (DNF) $\phi_1 \vee \cdots \vee \phi_m$ per a standard conversion. The result is potentially exponential in size; though this does not affect the data complexity, it may have a significant effect on performance in practice. However, this DNF conversion leads to further rewritings that may lead to better performance. First, we can express disjunctions using union ($\cup$) or using disjunctive ($\vee$) selection conditions. Second, since this expression falls on the right-hand side of an anti-semijoin operator, we can also express it as a sequence of such operators. Thus, for the translation of $(P - Q)^+$ into $P^+ \overline{\ltimes}_{\Uparrow} Q^?$, we can consider:

$$P^+ \overline{\ltimes}_{\Uparrow} Q^? = P^+ \overline{\ltimes}_{\Uparrow} \sigma_{\bigwedge_{1 \leq j \leq m} \theta_j}(Q'), \qquad \text{(CNF)}$$

$$P^+ \overline{\ltimes}_{\Uparrow} Q^? = P^+ \overline{\ltimes}_{\Uparrow} \sigma_{\bigvee_{1 \leq j \leq m} \phi_j}(Q'), \qquad \text{(DNF}_1\text{)}$$

$$P^+ \overline{\ltimes}_{\Uparrow} Q^? = P^+ \overline{\ltimes}_{\Uparrow} \bigcup_{1 \leq j \leq m} \sigma_{\phi_j}(Q'), \qquad \text{(DNF}_2\text{)}$$

$$P^+ \overline{\ltimes}_{\Uparrow} Q^? = P^+ \overline{\ltimes}_{\Uparrow} \sigma_{\phi_1}(Q') \ \ldots \overline{\ltimes}_{\Uparrow} \sigma_{\phi_m}(Q'). \qquad \text{(DNF}_3\text{)}$$

where Q' denotes the rewriting of join variables $\bar{X}$ in Q to produce Cartesian products on all join patterns and the subsequent application of $\beta_{\bar{X}, \bar{X_1}, \ldots, \bar{X_k}}$ to perform unification over those variables. Note, however, that in the cases of DNF$_2$ and DNF$_3$, some terms in the disjunction will *not* require a Cartesian product; for example, when we rewrite $P - (T_1 \bowtie T_2)$ to DNF, a disjunctive term on the right of the anti-semijoin will be $(T_1 \bowtie T_2)$ itself (the others will cover the case that join variables in T_1 or T_2 are bound to blanks). This suggests that these options *may* be more efficient despite a *potential* exponential blow-up.

Removing Explicit Unification: Given a base query of the form $P - Q$, if the join variables of Q do not correlate with P, we do not need to perform unification on them. Consider a query $Xpa - (XpY \bowtie Ypb)$. This can be rewritten to $Xpa \overline{\ltimes}_{\Uparrow} (\text{if}_{\,\text{bk}(Y_2), Y_1, Y_2, Y}(\sigma_{\theta_{\Uparrow}}(XpY_1 \bowtie Y_2pb)))$. However since Y does not appear on the left of the difference, we can simplify to $Xpa \overline{\ltimes}_{\Uparrow} (\sigma_{\theta_{\Uparrow}}(XpY_1 \bowtie Y_2pb))$.

Converting Anti-semijoins to Difference: Given a base query of the form $P - Q$, we can consider cases where the correlating variable(s) of P and Q may or may not yield blanks on either side. In particular, if Q returns a tuple with blanks for all correlating variables, then the entire difference $P - Q$ must be empty. On the other hand, if P returns a tuple with blanks for all correlating variables and Q is non-empty, then that tuple is removed from P. Finally, in cases where we know that the correlating variable(s) of P and Q cannot yield blanks[3], we can convert the anti-semijoin to standard difference. These ideas yield possible optimisations when we know more about which attributes can yield nulls.

[3] In standard relational settings, this might be if the correlating variables is a primary key of a table, for example. In RDF, we may detect such a case for subjects or objects of a given property that do not give blanks in a given dataset, for example.

Options for Difference: The SPARQL standard provides several ways for expressing difference. Here we consider two: the operators `MINUS` and `FILTER NOT EXISTS` (`FNE`). The SPARQL standard states that solutions of (P `MINUS` Q) are the solutions μ_1 of P such that there does not exist a solution μ_2 of Q where $\mathrm{dom}(\mu_1) \cap \mathrm{dom}(\nu_2)$ is not empty and μ_1 is joinable with μ_2. On the other hand, the solutions of P `FNE` Q are all solutions μ_1 of P such that there does not exist any solution μ_2 for $\mu_1(Q)$, where $\mu_1(Q)$ denotes the result of substituting in Q each variable X in $\mathrm{dom}(\mu_1)$ by $\mu_1(X)$. If $P - Q$ has at least one correlated variable, then P `MINUS` Q and P `FNE` Q are equivalent and can be interchanged.

6 Evaluation

Our evaluation presents an initial cost–benefit analysis of a certain answer semantics for SPARQL by addressing the following research questions: RQ1: How do the proposed SPARQL query rewriting strategies compare in terms of performance with the base query, with themselves, with similar results in an SQL setting, and for different SPARQL implementations? RQ2: Does a certain answer semantics significantly change query results in a real-world setting?

6.1 Evaluation Setting

In this section, we describe the SPARQL query engines selected, the machines and configurations used, as well as the datasets and queries. Supporting material can be found online: https://users.dcc.uchile.cl/~dhernand/revisiting-blanks.

Engines and Machines: The query rewriting strategy allows certain answers to be approximated on current SPARQL implementations. We test with two popular engines, with the added benefit of being able to cross-check that the solutions generated by both produce the same answers: Virtuoso (v.7.2.4.2) and Fuseki (v.2.6.0). The machine used is an AMD Opteron Processor 4122, 24 GB of RAM, and a single 240 GB Kingston SUV400S SSD disk; Virtuoso is set with `NumberOfBuffers` = 1360000 and with `MaxDirtyBuffers` = 1000000; Fuseki is initialised with 12 GB of Java heap space.

Rewriting Strategies: We consider various strategies: [`B`|`CNF`|$\mathtt{DNF}_{1,\ldots,3}$] where `B` denotes base queries, `CNF` queries in conjunctive normal form, and `DNF` queries in disjunctive normal form; we denote these variations as Γ in the following. [$\Gamma^{\nexists}$ | Γ^{-}] These queries use either `FNE` ($\nexists$) or `MINUS` ($-$) in SPARQL. [Γ|Γ^{*}] Rather than use `isBlank` to check if a node is blank or not, in case an engine cannot form an index lookup to satisfy such a condition, we also try adding a triple `(X,a,:Blank)` to the data for each blank `X` and a triple pattern to check for that triple in the query (denoted Γ^{*}); this does not apply to base queries. In total, this leads to 18 possible combinations. Rather than present results for all, we will highlight certain configurations in the results.

6.2 TPC–H Experiments

To address RQ1, we follow the experimental design of Guagliardo and Libkin [11] who provide experiments for PostgreSQL using the TPC-H benchmark. Their results compare the performance of approximations for certain answers with respect to four queries with negations. For this, they modified the TPC-H generator to produce nulls in non-primary-key columns with varying probabilities (1–5%) to generate more/less unknown values. They also use scale factors of 1, 3, 6, and 10, corresponding to PostgreSQL databases of size 1 GB, 3 GB, 6 GB and 10 GB, respectively. We follow their setting as closely as possible to facilitate comparison later. We wrote a conversion tool (similar to the Direct Mapping [4]) to represent TPC-H data as RDF, and convert the TPC-H SQL queries to SPARQL.

Unifications: We first evaluate the proposed rewriting strategies of unifications in the difference operator for SPARQL. The base format of the queries used is $P - (Q \bowtie R)$ where each P, Q, and R is a triple pattern. We then generate between 1,000 and 10,000 triples matching each triple pattern to perform tests at various scales. For the data matching the join variable on Q and R, we generate blanks with a rate of 1, 2, 4 and 8%. These experiments allow us to estimate the costs of unifications in difference without other query operators interfering.

Figure 1 presents performance results. For clarity, we present only a selection of configurations: CNF is equivalent to DNF$_1$ in this case and we only show the aforementioned $[\cdot^{\nexists}/\cdot^{*}]$ variations for the base query and DNF$_3$ (other variations performed analogously). The first row pertains to Virtuoso while the second pertains to Fuseki. All eight sub-plots are presented with log–log axes (base 10) at the same scale permitting direct comparison across plots (comparing horizontally across engines and comparing vertically across blank rates). The y-axis maximum represents a timeout of 25 min (reached in some cases by Fuseki).

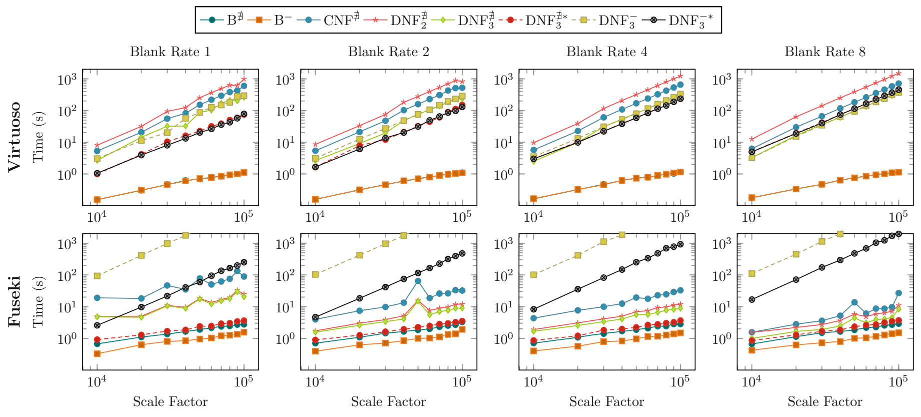

Fig. 1. Unification results for Virtuoso and Fuseki, varying scales and blank rates

(RQ1) The performance of the rewritten queries is (as could be expected) worse than the two base queries for all blank rates, scale factors and engines. In the case of Virtuoso, the base queries generally run in under one second; however, the fastest rewritten queries take at least a second and there is at least an order of magnitude difference between the base query and the fastest rewritten query. Looking at Fuseki, the fastest base query is slower than Virtuoso, but does generally tend to execute within one second (except at the larger scales). However, we see a number of rewriting strategies in the case of Fuseki where the difference is within half-an-order of magnitude of the fastest base case. Otherwise, we see that the choice of strategy is generally not sensitive to the blank rates considered (i.e., lines generally maintain the same ordering across plots), nor is it sensitive to scale (i.e., lines do not generally cross within plots).

Queries: The previous experiments looked at "atomic" unifications. We now run the four TPC-H queries used by Guagliardo and Libkin [11] considering a blank rate of 5%, four scale factors, and two engines. We employ a timeout of 10 min. We also choose one base query ($\mathrm{B}^{\not=}$) to be compared against the rewritten queries for approximating certain answers. Fuseki repeatedly times out for these experiments hence here we rather focus on the results of Virtuoso.

In Table 2, we present a comparison of the performance results for Virtuoso's fastest rewritten query and the results as presented by Guagliardo and Libkin [11]. More specifically, for a blank rate of 5%, the table shows the range of relative performance between the base query and the best rewritten query execution for that query; since Guagliardo and Libkin do not present absolute runtimes, our comparison is limited to relative performance. Note that due to differences in how SPARQL and SQL treat inequalities over nulls/blanks, Q_3 did not need rewriting for Virtuoso. For Q_2 in PostgreSQL, the actual results drop below the presented numeric precision, returning almost instantaneously for PostgreSQL once a null is found (which confirms that the results are empty).

(RQ1) We see that for Q_1, Virtuoso performs better in relative performance than PostgreSQL, for Q_2 PostgreSQL performs (much) better, for Q_3 there is little difference, while for Q_4 Virtuoso initially performs better than PostgreSQL but then at $\mathbf{SF} \geq \mathbf{3}$, Virtuoso begins to throw an error stating that an internal limit of 2097151 results has been reached (we could not resolve this). Aside from this latter issue, these results show that Virtuoso with our rewriting strategies is competitive with PostgreSQL under SQL-based rewritings for relative performance between base and rewritten queries. Furthermore, unlike in the previous experiments, we observe that in the case of Q_1 and Q_2, Virtuoso is now sometimes faster for the rewritten queries than the base queries: by removing uncertain answers, the number of intermediary solutions to be processed is reduced.

(RQ2) We observe three of the four base queries returning uncertain answers in SPARQL that do not hold under some valuations: for Q_1, 59% of answers are uncertain; for Q_2, all answers are uncertain; whilst for Q_4, 7% of answers are uncertain; we further highlight that these results are present for a blank rate of 5%. These results suggest that for queries with negation, evaluation under standard SPARQL semantics may in some cases return a significant ratio of uncertain/unsound answers even for modest levels of blanks in the dataset; this is to be expected given that, e.g., even a single blank tuple returned from the right-side of a difference can render all results uncertain (as per Q_2).

Table 2. Ranges of average relative performance for scale factor (SF) 1, 3, 6 and 10 on a fixed blank rate of 5%.

Q.	SF = 1	SF = 3	SF = 6	SF = 10
Virtuoso				
Q_1	0.95–0.96	0.95–0.96	0.97–0.99	0.94–0.95
Q_2	0.76–1.07	0.73–0.99	0.89–1.06	0.55–0.77
Q_3	1.00–1.00	1.00–1.00	1.00–1.00	1.00–1.00
Q_4	1.55–1.56	error	error	error
PostgreSQL (G&L [11])				
Q_1	1.01–1.03	0.99–1.01	0.98–1.01	1.00–1.02
Q_2	0.00–0.00	0.00–0.00	0.00–0.00	0.00–0.00
Q_3	1.01–1.04	1.01–1.04	0.99–1.02	1.00–1.06
Q_4	1.75–1.86	1.80–1.93	2.05–2.25	3.54–3.89

Table 3. Numbers of Wikidata use-case queries (from a total of 446) that could be affected by a certain answer semantics

Feature	A	B	C	D
`MINUS`	13	9	9	2
`FILTER NOT EXISTS`	23	15	10	1
`OPTIONAL` w/`!BOUND`	5	1	0	0
`!=`	7	5	3	0
Total	47	29	21	3

6.3 Wikidata Survey

Since the previous experiments are based on a synthetic benchmark converted from a relational setting, we performed an analysis of the user-contributed SPARQL queries on the Wikidata Query Service, which offers a more native Semantic Web setting[4]. As previously described, Wikidata uses blanks to represent unknown values; our goal now is to determine whether or not a choice of certain answer semantics could impact a current, real-world setting.

(RQ2) We first inspect the 446 queries to see which could potentially be affected by a certain answer semantics. We provide a summary in Table 3 according to the query features that may cause uncertain answers, with columns helping to indicate why queries with such features do not give uncertain answers in this context: **A** applies no assumptions, counting queries using the pertinent feature; **B** counts the queries that could still give uncertain answers knowing that Wikidata only uses blanks in a single object position; **C** counts the queries that could give uncertain answers further knowing which predicates have blanks; finally, **D** counts the queries whose solutions do change under certain answers. Hence, we see that 10.5% of the queries contain features that could cause uncertain answers, 6.5% of queries could cause uncertain answers even though Wikidata only uses blanks in a single object position, 4.7% of queries could cause uncertain answers knowing which predicates have blank values and which do not, while finally 0.6% of queries actually return uncertain answers.

We provide some statistics on the three Wikidata queries generating uncertain answers in Table 4. First for performance, we run the original query (T_1)

[4] https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples.

Table 4. Query execution times (ms) for the three Wikidata queries with uncertain answers, and ratio of uncertain answers to total answers

	Local (Virtuoso)			Public (Blazegraph)			Uncertain/total
	T_1	T_2	T_2/T_1	T_1	T_2	T_2/T_1	
Q_1	144464	142989	0.99	53012	TO	–	20/5487
Q_2	7038	521	0.07	1013	2045	2.02	42/42
Q_3	1266326	1419269	1.12	TO	TO	–	12/27221

and a rewritten version for certain answers (T_2) over both a local Virtuoso index of Wikidata, as well as the public Wikidata Query Service (running Blazegraph). (RQ1) While the performance of the first query is comparable under both standard and certain answer semantics for Virtuoso, the latter times out on Blazegraph. On the other hand, the second query is faster on Virtuoso for certain answer semantics, possibly because it is anticipated that all answers will be discarded. This is not the case for Blazegraph, where the rewritten query takes twice the time. In Virtuoso Q_3 takes slightly longer in the rewritten query. Blazegraph times out in all runs of Q_3. We also look at the ratio of uncertain answers the queries would return. (RQ2) The ratio for Q_1 and Q_3 are relatively low, but on the other hand, for Q_2, the ratio is 100%: all answers are uncertain.

For space reasons, we refer to the webpage for further analysis of the Wikidata queries, including more details on the queries returning uncertain answers.

7 Conclusions

In this paper, we have looked at the semantics of SPARQL with respect to RDF graphs that use blank nodes as existential variables encoding unknown values. In particular, we have investigated the feasibility of approximating certain answers in SPARQL, proposing various rewriting strategies. Our initial results suggest that querying for certain/possible answers generally does incur a significant cost, but that at least for Virtuoso, query answering is still feasible (and in some cases faster than under standard semantics). We showed that the relative performance results for Virtuoso under certain answer semantics are competitive with results published for PostgreSQL. In general, we saw that although some queries are executed faster under certain semantics with current SPARQL implementations, for others there can be a significant performance cost. It is important to highlight, however, that experiments were run using off-the-shelf SPARQL implementations; dedicated SPARQL implementations for approximating certain answers may further improve on the performance observed here.

Regarding the question of whether or not offering users a choice of certain answer semantics is important, we performed an analysis of 446 Wikidata queries, where although 10.5% use negation and inequality features that could cause uncertain answers in principle, only 0.6% of the queries return uncertain answers in practice. However, Wikidata only uses unique blanks (acting similar to unmarked nulls) in the object position. It would be interesting to do similar studies for other datasets using existential blanks, though we are not immediately aware of such a dataset that has a set of SPARQL queries to analyse.

In summary, though the results here confirm that certain answers can be effectively approximated using even off-the-shelf SPARQL implementations, the practical motivation for such a SPARQL semantics remains speculative.

Acknowledgements. The work was also supported by the Millennium Institute for Foundational Research on Data (IMFD) and by Fondecyt Grant No. 1181896.

References

1. Abiteboul, S., Kanellakis, P.C., Grahne, G.: On the representation and querying of sets of possible worlds. Theor. Comput. Sci. **78**(1), 158–187 (1991)
2. Ahmetaj, S., Fischl, W., Pichler, R., Simkus, M., Skritek, S.: Towards reconciling SPARQL and certain answers. In: World Wide Web (WWW), pp. 23–33 (2015)
3. Angles, R., Gutierrez, C.: The multiset semantics of SPARQL patterns. In: Groth, P., et al. (eds.) ISWC 2016. LNCS, vol. 9981, pp. 20–36. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46523-4_2
4. Arenas, M., Bertails, A., Prud'hommeaux, E., Sequeda, J.: A direct mapping of relational data to RDF. W3C Recommendation (2012)
5. Arenas, M., Pérez, J.: Querying semantic web data with SPARQL. In: Principles of Database Systems (PODS), pp. 305–316. ACM (2011)
6. Arenas, M., Ugarte, M.: Designing a query language for RDF: marrying open and closed worlds. In: Principles of Database Systems (PODS), pp. 225–236. ACM (2016)
7. Codd, E.F.: Understanding relations. SIGMOD Rec. **6**(3), 40–42 (1974)
8. Console, M., Guagliardo, P., Libkin, L.: Approximations and refinements of certain answers via many-valued logics. In: Knowledge Representation and Reasoning (KR), pp. 349–358. AAAI Press (2016)
9. David, C., Libkin, L., Murlak, F.: Certain answers for XML queries. In: Principles of Database Systems (PODS), pp. 191–202. ACM (2010)
10. Gheerbrant, A., Libkin, L., Tan, T.: On the complexity of query answering over incomplete XML documents. In: International Conference on Database Theory (ICDT), pp. 169–181 (2012)
11. Guagliardo, P., Libkin, L.: Making SQL queries correct on incomplete databases: a feasibility study. In: Principles of Database Systems (PODS), pp. 211–223. ACM (2016)
12. Guagliardo, P., Libkin, L.: Correctness of SQL queries on databases with nulls. SIGMOD Rec. **46**(3), 5–16 (2017)
13. Harris, S., Seaborne, A., Prud'hommeaux, E.: SPARQL 1.1 query language. W3C Recommendation, March 2013
14. Hayes, P., Patel-Schneider, P.F.: RDF 1.1 semantics. W3C Recommendation, February 2014
15. Hogan, A., Arenas, M., Mallea, A., Polleres, A.: Everything you always wanted to know about blank nodes. J. Web Sem. **27**, 42–69 (2014)
16. Klein, H.-J.: On the use of marked nulls for the evaluation of queries against incomplete relational databases. In: Workshop on Foundations of Models and Languages for Data and Objects. Kluwer (1998)

17. Libkin, L.: Certain answers as objects and knowledge. In: Knowledge Representation and Reasoning (KR). AAAI Press (2014)
18. Libkin, L.: SQL's three-valued logic and certain answers. In: International Conference on Database Theory (ICDT), pp. 94–109 (2015)
19. Libkin, L.: SQL's three-valued logic and certain answers. ACM Trans. Database Syst. **41**(1), 1:1–1:28 (2016)
20. Lipski Jr., W.: On relational algebra with marked nulls preliminary version. In: Principles of Database Systems (PODS), pp. 201–203. ACM (1984)
21. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. ACM Trans. Database Syst. **34**(3), 16:1–16:45 (2009)
22. Vrandecic, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Commun. ACM **57**(10), 78–85 (2014)