Query Answering over Incomplete and Uncertain RDF

Anastasios Kementsietsidis
IBM Research
tasosk@ca.ibm.com

Enela Pema
UC Santa Cruz
epema@cs.ucsc.edu

Wang-Chiew Tan
UC Santa Cruz			
tan@cs.ucsc.edu

ABSTRACT

While incompleteness and uncertainty naturally arise in real-world RDF data, the RDF model itself provides little support for incomplete and uncertain data. In this paper, we introduce practical extensions of the RDF model to represent incompleteness and/or uncertainty. We adopt the semantics of certain answers as the meaningful query answers in the presence of incompleteness and uncertainty, and we investigate the computational complexity of computing the certain answers to several fragments of SPARQL under our models. We determine that the problem is hard in general, and motivated by this intractability result, we develop a heuristic algorithm for query answering in the presence of uncertain data. Our algorithm can be implemented over any RDF query evaluation engine, and it can potentially be extended to handle both incompleteness and uncertainty together.

Keywords
RDF, uncertainty, incompleteness, blank nodes, certain answers, possible worlds

1. INTRODUCTION

When dealing with real data, it is seldom the case that we have all the data for the domain of discourse. Indeed, real datasets are inherently incomplete [4]. To exemplify, consider the set of Persons entities from the DBPedia [3] dataset, consisting of 709,703 persons. Even though we know that every person is born somewhere and on some date, the DBPedia dataset includes the birthdate and birthplace of only 420,242 and 323,368 persons, respectively. While one may not wish to model the unknown birth-date and birthplace of every single person, there are situations where we wish to express that certain properties are present, yet their values are not known a priori. Hence, although the birth-date of Aristotle is unknown to DBPedia, it would be more accurate to express the fact that Aristotle does indeed have a birth-date property with an unknown value.

Starting from the standard RDF model [13], a contribution of our work is an extension of RDF to incorporate incompleteness and uncertainty. An RDF dataset is a set of triples, each of the form (subject predicate object). Our RDF\textsubscript{c} extension allows a triple of the form: (Aristotle birthDate ?v), where ?v is a variable that represents the fact that Aristotle does have a birth date, but its precise value is unknown. We also consider scenarios in which the value is not completely unknown, but there is uncertainty regarding the actual value. We capture this form of uncertainty by associating a set of mutually exclusive alternatives to a variable in a RDF\textsubscript{c} dataset. For example, consider Plato, a Greek philosopher. Plato’s date of birth is 423B.C., 424B.C., 427B.C., or 428B.C., depending on the consulted source\textsuperscript{1}. Our RDF\textsubscript{c\textsuperscript{u}} extension allows an expression of the form: (Plato birthDate ?v) with an associated function \( \rho \), defined as: \( \rho(v) = \{423B.C., 424B.C., 427B.C., 428B.C.\} \). Here, ?v is a restricted variable whose restriction is defined through the function \( \rho \) on ?v. The restricted variable ?v denotes that Plato has an uncertain birth date that can take exactly one of the values in the set given by \( \rho(v) \). Together with \( \rho \), the triple above represents four possible worlds: (Plato birthDate 423B.C.), (Plato birthDate 424B.C.), (Plato birthDate 427B.C.), and (Plato birthDate 428B.C.). Uncertainty typically occurs when data is integrated from multiple sources which may disagree on what the value of a property should be. Instead of deciding upfront which value is correct, our model allows one to capture all alternative values. Observe that since \( \rho \) may not be defined on all variables in RDF\textsubscript{c\textsuperscript{u}}, the RDF\textsubscript{c\textsuperscript{u}} model captures both incompleteness and uncertainty.

Beyond modeling issues, it is important to investigate the right semantics for query answering in an RDF dataset that contains both incompleteness and uncertainty. Related works in this area (e.g., [9, 12, 16]) have largely devised their own semantics for query answering. Here, we adopt the notion of certain answers as the “meaningful” query answers that should be returned on incomplete and uncertain datasets. The certain answers are those query answers that can be derived from every possible world. The concept of certain answers is widely used in the context of incomplete and uncertain relational databases [7], data integration [8], and data exchange [5]. It has been generally regarded as the right concept for answering queries over many possible worlds. More precisely, if \( q \) is a query over an RDF dataset \( D \) that allows for both incompleteness and uncertainty, the certain answers of \( q \) on \( D \) are the answers found in the intersection \( \bigcap q(D') \), where \( D' \) ranges over all possible worlds of \( D \), and \( q(D') \) are the answers to \( q \) on \( D' \).


\textsuperscript{*}Copyright is held by the author/owner. Seventeenth International Workshop on the Web and Databases (WebDB 2014). June 22, 2014 - Snowbird, UT, USA.
A natural question is whether we can compute the certain answers of a query over incomplete and uncertain RDF data efficiently. To this end, we systematically analyze the theoretical complexity of computing certain answers of queries over different extensions of RDF models. Among our results, we show that the problem of computing the certain answers of a class of SPARQL\(^2\) queries, called conjunctive SPARQL queries, over RDF\(_V^o\) datasets is coNP-complete in general. The problem remains coNP-complete even when we restrict RDF\(_V^o\) datasets to contain uncertainty only among nodes that hold literal values. In light of the intractability result, we develop a heuristic algorithm for computing the certain answers to conjunctive SPARQL queries with FILTER conditions over RDF datasets where uncertainty can occur among literal nodes.

In computing the certain answers, our algorithm relies on a "representation graph" which holds all possible values associated to each uncertain node in a given dataset. To check if a tuple is a certain answer, we construct and evaluate a SPARQL query on top of the representation graph. Our algorithm can be implemented on top of any RDF query evaluation engine. We plan to experimentally evaluate our algorithm on realistic and synthetic RDF datasets containing uncertainty. We have high expectations regarding the efficiency of our algorithm since it significantly relies on the underlying RDF store while computing the certain answers. We also plan to extend our algorithm to handle RDF datasets containing both uncertainty and incompleteness.

**Related Work** Capturing and reasoning with incompleteness and uncertainty has gained significant attention in the RDF community. The W3C standard\(^1\) introduces blank nodes as special URIs that represent unknown values. However, there is a well-documented confusion regarding the semantics of blank nodes\(^1\). In particular, according to the standard, two blank nodes could represent the same unknown (e.g., two twins share the same blank node to denote the same unknown birthday). In contrast, in the W3C standard, query engines treat distinct blank nodes as distinct values when evaluating SPARQL queries (e.g., the two identical blank nodes that denote the twins' birthdates are treated as distinct birthdates). As a result of this ambiguity, blank nodes are not typically used for representing incomplete information. Instead, they are typically used as special URI nodes to group a collection of related properties together.

OWL semantics can be used to model incomplete and uncertain information. For instance, (Aristotle birthDate ?e) can be modeled by adding a triple (Aristotle rdf:type C), where C is a new concept, defined as a subclass of the class expression someValuesFrom(birthDate, Thing). We can express uncertainty about ?e by defining a concept C to be a subclass of someValuesFrom(birthDate, oneOf(427 B.C., 428 B.C., 429 B.C.)). However, there are forms of incompleteness and uncertainty that can be captured by our model, but cannot be expressed in OWL. For example, we can express that two different people (e.g., twins) have the same birth date, by associating the same variable to both. However, this is not possible with OWL. In general, it is not possible to capture constraints with a non-tree like structure in OWL. While this is possible in our models, since a variable can be the object of more than one triple. Finally, the OWL semantics is meant to express complex constraints. Thus, reasoning with such constraints can be highly expensive as well. In our models, we focus on simple, yet useful forms of incompleteness and uncertainty that allow us to develop efficient heuristics for query answering.

Probabilistic models have been proposed for uncertainty management\(^{[9, 16]}\). By associating probabilities to nodes and edges of the RDF graph, one can in principle model uncertainty. However, probabilistic models would require one to specify probabilities on each of the alternative values that an unknown attribute can take. This requirement becomes infeasible if an unknown attribute may take values from a very large or infinite domain. Moreover, probabilities need to be maintained as the dataset changes. In contrast, our model seamlessly accommodates new data without such limitations. Another drawback of probabilistic models is that they rely on non-standard semantics for query answering. Furthermore, computation of probabilistic query answers can be expensive and most algorithms rely on approximation heuristics. As such, these algorithms are not suitable and more expensive than computing the certain answers.

In\(^{[12]}\), the authors propose URDF, a framework for uncertainty management, which uses consistency rules to express constraints on the data. These rules, in particular, can be used to capture uncertainty. However, their semantics of query answers does not coincide with certain answers.

**2. BACKGROUND AND PRELIMINARIES**

Our formalisms on RDF mostly follow that of\(^{[11, 14, 17]}\). In this paper, we focus on the simple RDF model, which excludes the RDF Schema. We assume we are given pairwise disjoint infinite sets of URIs (U), Blank nodes (B), and Literals (L). An RDF triple is a triple (s, p, o) \(\in (U \cup B) \times (U \cup B \cup L)\), where s is the subject, p the predicate, and o the object. An RDF dataset (also referred to as an RDF graph) is a set of RDF triples.

SPARQL is the standard query language for RDF, based on graph-pattern matching\(^{[15]}\). Here, we consider the following SPARQL fragments. Conjunctive SPARQL queries (denoted CQSPARQL), are SPARQL queries of the form:

\[
\text{SELECT}\ (\text{select list})
\]
\[
\text{WHERE}\ (\text{basic graph pattern})
\]

where (select list) is a list of constants or variables that appear in the graph pattern. The class CQSPARQL/ contains CQSPARQL queries extended with a \text{FILTER} condition. Specifically, CQSPARQL/ queries are of the form:

\[
\text{SELECT}\ (\text{select list})
\]
\[
\text{WHERE}\ (\text{basic graph pattern})
\]
\[
\text{FILTER}\ (\text{filter condition})
\]

where the filter condition is an expression of terms combined using the logical connectives \&\& and \|, with each term being of the form \(e_1 \text{ op } e_2\). Here, \(e_1\) and \(e_2\) are variables or literals and \(op\) is one of \(\_, \neq, \leq, >, \geq\). We denote by CQSPARQL\(^*\), queries in CQSPARQL/ in which the terms in the filter condition contain only inequalities. Finally, boolean SPARQL queries are SPARQL queries written using the \text{ASK} clause (instead of \text{SELECT}) to return boolean answers.

**3. INCOMPLETE AND UNCERTAIN RDF**

Here, we introduce variables to represent unknowns in RDF, whose value can range over U or L (in the case of incompleteness), or can be restricted to a finite set of alternatives from U or L (in the case of uncertainty). We formally define the models and discuss complexity in Section 3.1. In Section 3.2, we consider a variation of our model, where variables can only occur among literals.

**3.1 RDF with Variables**

Given pairwise disjoint infinite sets \(U, V,\) and \(L,\) of URIs, Variables and Literals, respectively, an RDF\(_V\) triple is a triple of the

\(^2\)The SPARQL query language is the W3C standard for querying RDF\(^{[15]}\).
form \((s, p, o) \in (U \cup V) \times U \times (U \cup V \cup L)\). An RDF\(_V\) dataset (or graph) is a set of RDF\(_V\) triples.

In RDF\(_V\), a variable is used if we have no knowledge of the missing property value (as was the case for Aristotle’s year of birth). However, in many situations, we have some knowledge regarding possible property values (as was the case for Plato’s year of birth).

To express partial knowledge in the form of alternative values, we introduce the notion of a variable domain function, which is a function that associates a finite set of possible values to a variable in the dataset. Let \(A = \mathcal{P}(U) \cup \mathcal{P}(L)\), where \(\mathcal{P}(U)\) and \(\mathcal{P}(L)\) are the sets of all finite non-empty subsets of, respectively, \(U\) and \(L\). A variable domain function \(p\) is of the form \(p : V^+ \to A\), where \(V^+ \subseteq V\). An RDF\(_V\) dataset is a pair \((G, p)\), where \(G\) is an RDF\(_V\) dataset and \(p\) is a variable domain function. Observe that the function \(p\) may not be defined over all variables in the dataset. Given an RDF\(_V\) dataset \((G, p)\), if \(v\) is a variable in \(G\) such that \(p\) is defined on \(v\), we call \(v\) a restricted variable.

We distinguish a special case of RDF\(_V\) datasets, when the function \(p\) is defined over all variables in \(G\), i.e., \(G\) has only restricted variables. In this case, we write \(\bar{p}\) instead of \(p\). We say that an RDF\(_V\) dataset \((G, \bar{p})\) is a pair \((G, \bar{p})\) where \(\bar{p}\) is a variable domain function that is defined over all variables in \(G\).

Given an RDF\(_V\) graph \(G\), it is easy to see that \(G\) gives rise to a set of possible worlds, where each possible world is an RDF dataset obtained from \(G\) via a total mapping \(h : V \to U \cup L\). We write \(h(G)\) to denote the dataset obtained from \(G\) by applying mapping \(h\) on each variable of \(G\). Obviously, if \(\forall v \in V\) and there exists a triple \((!v, p, o)\) in \(G\), then \(h(!v)\) is a triple in \(h(G)\). If \(\exists v \in V\) and \((s, p, v)\) is a triple in \(G\), then \(h(v)\) is a triple in \(U \cup L\). Clearly, there are infinitely many such mappings, which give rise to an infinite set of possible worlds. A standard notion of what constitutes the set of answers of a query over a database with variables is that of the certain answers [5], which we formally re-define here:

**Definition 1.** Let \((G, p)\) be an RDF\(_V\) dataset.

- The set of possible worlds of \(G\), denoted as \(\mathcal{PW}(G)\), is the set of all RDF datasets \(G'\) for which there exists a mapping \(h : V \to U \cup L\) such that \(h(G) = G'\) and for every restricted variable \(?v\) in \(G\), we have \(h(?v) \in p(?v)\).

- Let \(q\) be a SPARQL query. The certain answers of \(q\) on \(G\) is the set \(\bigcap_{G' \in \mathcal{PW}(G)} q(G')\).

In other words, the certain answers of \(q\) on \(G\) consist of all answers in the intersection of \(q(G')\), where \(G'\) denotes a possible world of \(G\). Since the set \(\mathcal{PW}(G)\) is possibly infinite, it is infeasible to compute \(\text{certain}(q)\) by constructing all possible worlds \(G'\) of \(G\) and taking the intersection of all \(q(G')\).

### Complexity of Query Answering over RDF\(_V\)

Let \(q\) be a fixed SPARQL query. Given an RDF\(_V\) dataset \((G, \bar{p})\) and a tuple \(t\), a basic decision problem associated with the computation of the certain answers is asking whether or not \(t\) is a certain answer of \(q\) over \((G, \bar{p})\). More precisely, \(\text{certain}(q)\) is the following decision problem: For a fixed SPARQL query \(q\), given an RDF\(_V\) dataset \((G, \bar{p})\) and a tuple \(t\), is \(t\) an answer to \(q\) in every possible world of \(G\)?

Observe that the decision problem formulation above captures the data complexity of computing the certain answers with respect to the size of the graph only, since \(q\) is fixed. Next, we present complexity results concerning \(\text{certain}(q)\) for various fragments of SPARQL, under the RDF\(_V\) model. We also consider two restricted cases of the RDF\(_V\) model. One restricted model is RDF\(_V\) (no function \(p\) is provided) which models incompleteness only, and the other is RDF\(_V\), which models uncertainty only.

The following proposition states a positive result concerning the certain answers of CQSPARQL queries over RDF\(_V\) graphs:

**Proposition 1.** Let \(q\) be a fixed CQSPARQL query. Then for any RDF\(_V\) dataset \(G\), the certain answers of \(q\) on \(G\) can be computed in polynomial time in the size of \(G\).

Proposition 1 follows immediately from [7], where they showed that the certain answers of conjunctive queries can be computed in polynomial time in the size of \(v\)-tables. By a similar token, the certain answers of a CQSPARQL query can be computed over the RDF\(_V\) dataset by treating variables as URIs or literals. The resulting answers that do not contain variables are returned as the certain answers of the query.

Next, we turn our attention to the CQSPARQL\(_q\) fragment. It was shown that the problem of computing the certain answers of conjunctive queries with inequalities over \(v\)-tables is coNP-hard [1, 5] in general. Although this result can be used to derive the hardness of computing the certain answers in our case, we give here an explicit reduction for a fixed SPARQL query with inequalities (see Theorem 1). Our proof of Theorem 1 relies on a polynomial reduction from Monotone 3SAT [6].

**Theorem 1.** Let \(q\) be the following CQSPARQL\(_q\) query:

\[
\text{ASK} \{ \text{?yp ?z1, ?yp ?z2, ?yp ?z3, ?yp ?z4} \\
\text{FILTER} \{ (?z1 \neq ?z2) \text{ AND } (?z2 \neq ?z3) \text{ AND } (?z1 \neq ?z3) \}
\]

Then, \(\text{certain}(q)\) is coNP-hard.

A straightforward corollary of Theorem 1 is that computing the certain answers to CQSPARQL\(_q\) queries over RDF\(_V\)\(_q\) datasets is coNP-hard in general.

Next, we consider the case where all variables in the dataset are restricted to only take values from a finite set of alternatives. It turns out that computing the certain answers to SPARQL queries over an RDF\(_V\)\(_q\) dataset is coNP-hard even for CQSPARQL\(_q\) queries.

**Theorem 2.** Let \(q\) be the fixed boolean CQSPARQL\(_q\) query ASK \{ ?x 1 ?y. ?z 0 ?y\}. Then, certain\((q)\) is coNP-hard.

This result is proven by giving a polynomial reduction from Monotone 3SAT. It follows from Theorem 2 that computing the certain answers to CQSPARQL\(_q\) queries over RDF\(_V\)\(_q\) datasets can also be coNP-hard in general.

### 3.2 RDF with Literal Variables

Consider the RDF\(_V\) model and a corresponding RDF\(_V\) dataset (similar observations apply for the RDF\(_V\)\(_q\) model and datasets). By definition, a variable in the dataset can be replaced by a URI. A possible world with a distinct graph structure is formed when all variables are replaced by URIs. For example, the simple RDF\(_V\) dataset shown in Figure 1(a) can have multiple possible worlds with different graph structures (see (b), (c), (d) in Figure 1). The replacement of variables with URIs leads to what we call structural incompleteness (uncertainty) since not only the content of nodes but also the structure of graphs is different between worlds. We contrast this form of incompleteness (uncertainty) from what we call literal incompleteness (uncertainty) in which variables are only replaced with literals. In the latter form, all possible worlds of the same RDF\(_V\) dataset have exactly the same structure since literal replacement does not change the internal graph structure. In Figure 1, if we focus on literal incompleteness, all possible worlds have the graph structure of Figure 1(a), and the only difference between them is the literal value of variable ?u. The complexity results from the previous section have motivated us to focus our attention to the simpler...
(and more common in practice) literal incompleteness and uncertainty. Furthermore, such form of incompleteness and uncertainty is commonly observed in real datasets [10].

We define now the literal incomplete and uncertain model \( \text{RDF}_\preceq \), in which variables appear only among object nodes, and a variable can only be associated with a set of alternatives that are all literals. Assume there are pairwise disjoint infinite sets \( U, V, \) and \( L \). Let \( A \) be the set of all finite and non-empty subsets of \( L \). An \( \text{RDF}_{\preceq} \) dataset is a pair \((G, \rho)\), where: (i) \( G \) is an \( \text{RDF}_\preceq \) dataset with triples \((s, p, o) \in U \times U \times (U \cup V \cup L)\), and (ii) \( \rho \) is a variable domain function of the form \( \rho: V \rightarrow A \). When no function \( \rho \) is provided, we say that \( G \) is an \( \text{RDF}_\preceq \) dataset. When \( \rho \) is defined on every variable of \( G \), we write \( \rho \) instead of \( \rho \), and we say that the pair \((G, \rho)\) is an \( \text{RDF}_{\preceq} \) dataset.

**Complexity of Query Answering over \( \text{RDF}_{\preceq} \).** We now look at the complexity of computing the certain answers over \( \text{RDF}_{\preceq} \) datasets. We consider \( \text{RDF}_\preceq \) and \( \text{RDF}_{\preceq} \) separately.

**Complexity for \( \text{RDF}_\preceq \):** A straightforward corollary of Proposition 1 points out that computing the certain answers of CQSPARQL queries over \( \text{RDF}_\preceq \) datasets can be done in PTIME. For the \( \text{RDF}_\preceq \) model, CQSPARQL queries with \( \text{FILTER} \) conditions with inequalities give rise to intractability (see Theorem 2). A similar proof of Theorem 1 where variables occur only among literal nodes can be used to establish coNP-hardness even under the \( \text{RDF}_\preceq \) model.

**Complexity for \( \text{RDF}_{\preceq} \):** To make matters worse, the complexity of computing the certain answers remains coNP-hard for CQSPARQL queries over \( \text{RDF}_{\preceq} \) datasets. This result can again be proven via a reduction from Monotone 3SAT. Table 1 summarizes the complexity results presented in this section.

<table>
<thead>
<tr>
<th>Complexity results</th>
<th>( \text{RDF}<em>\preceq ), ( \text{RDF}</em>{\preceq} )</th>
<th>( \text{RDF}<em>{\preceq}, \text{RDF}</em>{\preceq} )</th>
<th>( \text{RDF}<em>{\preceq}, \text{RDF}</em>{\preceq} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQSPARQL</td>
<td>PTIME</td>
<td>coNP-hard</td>
<td>coNP-hard</td>
</tr>
<tr>
<td>CQSPARQL/</td>
<td>coNP-hard</td>
<td>coNP-hard</td>
<td>coNP-hard</td>
</tr>
</tbody>
</table>

**Table 1: Complexity results**

**4. ALGORITHM FOR QUERY ANSWERING**

Given these hardness results, we develop a heuristic algorithm for computing the certain answers of CQSPARQL queries over \( \text{RDF}_{\preceq} \) datasets. The algorithm we will present computes exactly the certain answers, by applying heuristics that avoid computation of all possible worlds.

**Computing certain answers over \( \text{RDF}_{\preceq} \):** Before we describe our algorithm for computing the certain answers to CQSPARQL queries over \( \text{RDF}_{\preceq} \) datasets, we provide some intuition via a simple example.

**Example 1.** Let \((G, \rho)\) be the \( \text{RDF}_{\preceq} \) dataset of Figure 2. Let \( q \) be the CQSPARQL query in Figure 2, asking whether there is a scientist who is born in the same year as some philosopher.

The algorithm starts by determining the potential (or candidate) certain answers. These are the answers to \( q \) that have a chance of being true in every possible world. Together with each potential answer, we collect all mappings that could give rise to that potential answer in some possible world. We do this by using another query \( q_t \) constructed from \( q \). The query \( q_t \) in Figure 2 is constructed as follows: We relax the \( \text{FILTER} \) condition to not check if \( ?o_1 = ?o_2 \). To accept mappings of the graph pattern to \( G \) that might give rise to a potential answer in some possible world, we then add variables \( ?o_1 \) and \( ?o_2 \) in the select list. The answers to \( q_t \) over \( G \) are shown in the table in Figure 2. The only potential answers are \( t_1 = \langle \text{Albert_Einstein} \rangle \) and \( t_2 = \langle \text{Daniel_Bernoulli} \rangle \). The result of \( q_t \) also provides us with useful information about mappings that might give rise to a particular potential answer. For example, tuple \( s_1 \) indicates that \( \langle \text{Albert_Einstein} \rangle \) can be a query answer in a possible world where \( ?u_1 = \text{1879} \). For each potential answer, we use the result of \( q_t \) and the graph \( G_\rho \) to determine if there exists a possible world in which a particular potential answer is not found. Intuitively, \( \langle \text{Albert_Einstein} \rangle \) will not be a certain answer if there exists a possible world obtained by assigning to \( ?u_1 \) a value that is different from \text{1879} and \text{1880}. We do this verification for each potential answer via a check query, which is an ASK query that when evaluated over \( G_\rho \), it returns true if and only if the potential answer is false in some possible world. The check query \( q_{c_1} \) is constructed as follows: For every \( ?u_1 \in V \), its graph pattern contains a triple of the form \((?u_1)\text{ (hasValue) } \langle x_1 \rangle \). Specifically, the graph pattern of \( q_{c_1} \) will contain only \((?u_1)\text{ (hasValue) } \langle x_1 \rangle \). Next, we “guard” the \( \text{FILTER} \) condition for each tuples \( s \in q_t(G) \). From \( s_1 \) we obtain \( ?x_1 \neq \text{1879} \) and from \( s_2 \) we obtain \( ?x_1 \neq \text{1880} \). We take the conjunction of the two, and add it as a \( \text{FILTER} \) condition to \( q_{c_1} \). Finally, the check query \( q_{c_1} \):\n
\[
\text{ASK} \{ \langle x_1 \rangle \text{ (hasValue) } \langle x_1 \rangle \text{ FILTER ((?x_1 \neq \text{1879}) \& \& (?x_1 \neq \text{1880}))} \}
\]

Obviously, \( q_{c_1}(G_\rho) \) is false. Hence, \( t_1 \) is a certain answer. Similarly, we can construct a check query for \( t_2 \). Since there are no variables in \( s_3 \) and \( s_4 \), the check query \( q_{c_2} \) is: \( \text{ASK} \{ \langle u_1 \rangle \text{ (hasValue) } \langle x_1 \rangle \text{ FILTER true} \} \). Trivially, \( q_{c_2} \) is true and therefore \( t_2 \) is not a certain answer.

Our algorithm, called \( \text{CERTAINANSWERS}_\rho \), is described in Figure 4. The algorithm takes two inputs: an \( \text{RDF}_{\preceq} \) dataset \((G, \rho)\), and a CQSPARQL query \( q \). To simplify the presentation of our algorithm, without loss of generality, we make the following assumptions about \( q \): 1) every variable in the graph pattern of \( q \) appears without repetitions, and 2) the object of every triple in the graph pattern is a variable. Any CQSPARQL query can be rewritten to satisfy these properties. If \( ?o \) appears multiple times in the
We first compute a dataset \( \mathcal{G} \), to represent the set of alternative values assigned to each variable by \( \rho \). More specifically, \( \mathcal{G} \) is the set of triples \((\tau, v, \langle \)hasValue\rangle\), where \( v \) is a variable in \( G \), \( \langle \)hasValue\rangle is a special URI not used in \( G \), and \( \langle \)hasValue\rangle \( \in \rho(\tau) \). Internally, we represent variables in \( G \) as URIs, and associate to them a type “variable”.

**Figure 2:** Top: An example RDF graph \((G, \rho)\) where \( \rho \) is encoded as an RDF dataset (graph on the right). Bottom: Input query \( q \) to Algorithm CERTAINANSWERS\(_{\rho}^{q}\); the modified query \( q_{f} \) that is obtained in Phase 2. The result of \( q_{f} \) on \( G \).

**Figure 3:** The query \( q_{2} \) is normalized from \( q_{1} \).

Example 1 illustrates the main steps behind our algorithm, CERTAINANSWERS\(_{\rho}^{q}\), which we describe next. Algorithm CERTAINANSWERS\(_{\rho}^{q}\) runs in three main phases.

**Phase 0:** We first compute a dataset \( \mathcal{G}_{\rho} \), to represent the set of alternative values assigned to each variable by \( \rho \). More specifically, \( \mathcal{G}_{\rho} \) is the set of triples \((\langle \)hasValue\rangle, \langle \)hasValue\rangle, \langle \)hasValue\rangle\), where \( v \) is a variable in \( G \), \( \langle \)hasValue\rangle is a special URI not used in \( G \), and \( \langle \)hasValue\rangle \( \in \rho(v) \). Internally, we represent variables in \( G \) as URIs, and associate to them a type “variable”.

**Phase 1:** In this phase, we compute a set of potential answers, together with all mappings that may give rise to these potential answers in some possible world of \( G \). We describe this phase at a high level. In general, if \( \mu \) is a mapping from \( \mathcal{G}_{\rho} \) to \( G \), then \( \mu \) gives rise to a query answer if it satisfies \( \phi \) as well. However, when \( G \) contains variables, \( \mu \) may map variables of \( q \) to variables in \( G \). In this case, while \( \mu \) is always a mapping of \( \mathcal{G}_{\rho} \) to every possible world of \( G \), it may be the case that \( \phi \) is not satisfied in every possible world. For this reason, for a potential answer \( t \), we collect all mappings of \( \mathcal{G}_{\rho} \) to \( G \) that give rise to \( t \), when the filter condition is relaxed to always be satisfied. This is achieved via the query \( q_{f} \), obtained by transforming \( q \). The transformation only concerns *relevant variables*, which are variables that occur in \( \phi \). Query \( q_{f} \), in addition to variables from select_list, it also returns all relevant variables. Next, to ensure that \( q_{f}(G) \) will return all possible mappings that might give rise to a potential certain answer in some possible world of \( G \), we relax the filter condition \( \phi \) by replacing with *true* all terms that mention some relevant variable.

**Phase 2:** In this phase, we use the tuples in \( q_{f}(G) \) to construct a check query \( q_{c} \) for every potential answer \( t \). Each query \( q_{c} \) is evaluated on \( \mathcal{G}_{\rho} \) to determine whether or not there is an assignment to the variables in \( G \) that gives rise to a possible world \( G_{v} \), such that \( t \notin q(G_{v}) \). The tuple \( t \) is a certain answer if and only if no such assignment is found. For a potential answer \( t \), the set MAPPINGS\(_{t} \) contains tuples \( s \), where each \( s \) represents a mapping \( \mu_{s} \) of \( \mathcal{G}_{\rho} \) to \( G \) that may give rise to \( t \) in the query answers of some possible world. The purpose of Phase 2 is to check if there exists a possible world such that none of the mappings \( \mu_{s} \) satisfies the FILTER condition, which suffices to determine that \( t \) is not a certain answer. This check is done by evaluating a query \( q_{c}(G_{v}) \). The graph pattern of \( q_{c} \) consists of triples \( \langle s, \langle \)hasValue\rangle, \langle \)hasValue\rangle, \langle \)hasValue\rangle\rangle \( \forall v \in \mathcal{V} \). Notice that a mapping \( hc \) of \( \mathcal{G}_{\rho} \) to \( G_{v} \) represents a possible assignment of variables \( v \) to literals; thus, representing a possible world of \( G \). The purpose of \( \psi \) is to check that in a possible world, none of the mappings \( \mu_{s} \) is satisfied. Given a tuple \( s \in \text{MAPPINGS} \( \{ t \} \) \( \psi \) checks if \( \psi(\mu_{s}(t)) \) is true by first grounding \( \phi \) with \( s \), i.e., substituting each variable in \( \phi \) with the value assigned to it in \( s \) and then substituting each \( v \) with \( \psi(\mu_{s}(t)) \) \( \psi(\mu_{s}(\langle \)hasValue\rangle) \). Therefore, if \( \psi \) evaluates to true in some possible world, that means that \( t \) is not a certain answer; otherwise \( t \) is a certain answer.

**Theorem 3.** Given a CQSPARQL\(_{\rho}^{q}\) query \( q \) and RDF graph \((G, \rho)\), algorithm CERTAINANSWERS\(_{\rho}^{q}\) computes the certain answers of \( q \) on \((G, \rho)\).

**Analysis of the algorithm and optimizations** In Algorithm CERTAINANSWERS\(_{\rho}^{q}\), we construct a check query for every potential answer. Obviously, the number of potential answers is polynomial in the size of \( G \); hence, polynomially many check queries are constructed and evaluated. Moreover, for each potential answer, the corresponding check query is constructed based on the output of \( q_{f} \) on \( G \). A check query has as many triples as there are variables in \( G \). Also, the size of the formula \( \psi \) in the FILTER condition is a polynomial in the size of \( q_{f}(G) \). Evaluation of CQSPARQL\(_{\rho}^{q}\) queries, is exponential in the size of the dataset and the query together [2]. Since the size of the check queries depends on the size of \( G \), the evaluation of the check queries is exponential in the size of \( G \).

Our strategy in Algorithm CERTAINANSWERS\(_{\rho}^{q}\) has been to rely mainly on the query engine to compute the certain answers. While a check query can become large because its size depends on the size of \( G \), we expect check queries to be evaluated really fast by SPARQL query engines. The reason is because these queries are...
to be evaluated over the rather small dataset $G_p$, and moreover, the structure of $G_p$ and the structure of the graph pattern of the check query is quite simple. However, one potential downside of our algorithm can be the fact that the number of check queries to be evaluated depends on the output of $q_s$, which in general can be quite large. For this reason, we are planning to implement a few optimizations to reduce the result set of $q_s$. First, given a RDF$_{CV}$ dataset, it is natural to assume that some certain answers may be derived from the part of the dataset containing no variables. Therefore, one important optimization we plan to implement is to compute a subset of the certain answers from the variable-free triples in $G$. In fact, it is possible to come up with a SPARQL query $q_{certain}$ which, when evaluated over the dataset, in the usual sense of query evaluation (treating variables as constants), returns a subset of the certain answers. This optimization could preclude PHASE 1 of Algorithm CERTAINANSWERS$_{CV}$. Subsequently, the query $q_s$ can be written to avoid collecting mappings that give rise to query answers that have already been found to be certain by $q_{certain}$. Additionally, as we construct $q_s$, it is not necessary to relax all FILTER conditions. For instance, assume there is a condition $?o_1 = ?o_2$ in the FILTER. Assume also that $?o_1$ and $?o_2$ appear only as subjects in the graph pattern of $q$. Then, we already know that $?o_1$ and $?o_2$ will never be mapped to a variable in $G$. Thus, it is not necessary to relax the condition, and use check queries to determine if it is not satisfied in some possible world.

5. CONCLUSION AND FUTURE WORK

We have described simple and practical extensions of RDF for capturing incompleteness and/or uncertainty, and also investigated the computational complexity of certain query answering. We show that computing the certain answers of CQSPARQL queries over RDF datasets with uncertainty can be coNP-hard in data complexity, even when uncertainty is restricted to literal nodes. Motivated by this intractability result, we develop a heuristic algorithm for computing the certain answers for CQSPARQL queries over RDF datasets where uncertainty can occur among literal nodes. We emphasize that an advantage of our algorithm is that it enables one to leverage any off-the-shelf RDF store and query engine that may provide for improved performance of SPARQL query evaluation.

We plan to implement and test our algorithm over RDF stores/query engines, such as TDB and Virtuoso. In addition, we will implement several optimizations, with the goal to minimize the amount of work done to check potential certain answers. Furthermore, we plan to adapt Algorithm CERTAINANSWERS$_{CV}$ to compute the certain answers on RDF$_{CV}$ datasets, i.e., datasets that contain both incompleteness and uncertainty.

6. ACKNOWLEDGMENTS

This work was supported in part by NSF grant IIS-0905276, a Google Faculty Award, and an IBM Faculty Award.

7. REFERENCES