Expressing No-Value Information in RDF

Fariz Darari, Radityo Eko Prasojo, and Werner Nutt

Faculty of Computer Science, Free University of Bozen-Bolzano, Italy
{fariz.darari,radityoeko.prasojo}@stud-inf.unibz.it
nutt@inf.unibz.it

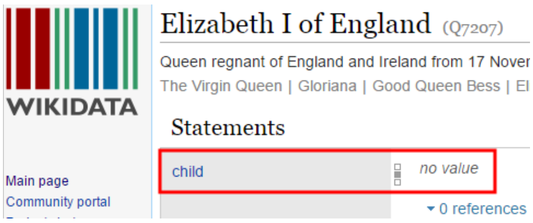
Abstract. RDF is a data model to represent positive information. Consequently, it is not clear how to represent the non-existence of information in RDF. We present a technique to express such information in RDF and incorporate it into SPARQL query answering. Given an empty query answer, our technique can distinguish whether it is empty due to possibly incomplete information, or non-existent information.

Keywords: RDF, negative knowledge, nulls, SPARQL

Reasons to visit: To discover how no-value information can (1) be represented in RDF and (2) be used to infer SPARQL query emptiness.

1 Introduction

RDF is mainly used to express positive information. However, representing negative information is often of interest in practice. For instance, on Wikidata [1], we have the following information about Elizabeth I not having any children.



Elizabeth I of England (Q7207)

Queen regnant of England and Ireland from 17 Nover

The Virgin Queen | Gloriana | Good Queen Bess | El

Statements

child — no value

0 references

Fig. 1. No-value information on Wikidata (http://j.mp/elizabethI)

In the above figure, Wikidata explicitly states that Elizabeth I had no children since the property `child` has "no value".[1] This is different than not recording anything at all which would imply possibly incomplete information for the children of Elizabeth I. To express this in RDF, one may be tempted to assign a special datatype constant `noValue` to represent the no-value information of the children of Elizabeth I, creating the triple $(\mathit{elizabethI}, \mathit{child}, \mathit{noValue})$. However, this creates a problem since executing the SPARQL `ASK` query $Q = (\{\}, \{(\mathit{elizabethI}, \mathit{child}, ?y)\})$ asking if Elizabeth I has a child, would give the answer 'yes'. Indeed, due to no formal definition, it is not clear how to properly use `noValue`.

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

The notion of no-value information was first introduced in the relational databases [2]. There, the term 'null value' was used, which may have different meanings: there exists no value (i.e., non-existence); there exists a value but it is unknown; or it is unknown whether a value exists. For the second case, we can leverage RDF blank nodes, while for the third case, the Open World Assumption (OWA) of RDF simply permits it. However, RDF cannot represent the first case, which is the no-value nulls, while in fact this no-value information is useful to distinguish from incomplete information. Furthermore, by having no-value information, an empty query answer can have two different meanings: whether it is empty because of possibly incomplete information, or whether it is truly empty from information that does not exist in the real-world.

In this poster, we present a technique for representing no-value information in RDF and incorporating such information into query answering. This introduction is followed by a formalization of no-value information and query answering in the presence of such information, a concrete RDF representation of no-value information, and a discussion.

2 Formalization

Preliminaries. Assume there are three pairwise disjoint infinite sets I (*IRIs*), L (*literals*) and V (*variables*). A tuple $(s,p,o) \in I \times I \times (I \cup L)$ is called a triple. An *RDF graph* G consists of a finite set of triples.

SPARQL is the standard query language for RDF [3]. The basic building blocks of a SPARQL query are triple patterns, which look like RDF triples, except that in each position, variables are also allowed. In this work, we focus on the conjunctive fragment of SPARQL where queries are represented as *basic graph patterns* (BGPs), that is, sets of triple patterns. The evaluation of a BGP P over G is defined as $[\![P]\!]_G = \{\, \mu \mid \mu P \subseteq G \text{ and } dom(\mu) = var(P) \,\}$. Given a query $Q = (W, P)$, where P is a BGP and $W \subseteq var(P)$ is the set of distinguished variables, the evaluation $[\![Q]\!]_G$ is the restriction of $[\![P]\!]_G$ to W. Over Q, we define the *prototypical graph* $\tilde{P}$ as the graph resulting from mapping each variable in P to a fresh IRI. The prototypical graph encodes any possible graph that can satisfy the query. Furthermore, a `CONSTRUCT` query has the abstract form (`CONSTRUCT` P_1 P_2) where both P_1 and P_2 are BGPs. Evaluating a `CONSTRUCT` query over G in a graph where P_1 is instantiated with all the mappings in $[\![P_2]\!]_G$.

Let us now formalize no-value information. We first define no-value statements to capture which information is non-existent.

Definition 1 (No-Value Statement). *A* no-value statement N *is defined as* $No(P)$ *where* P *is a BGP. To* N*, we associate the* `CONSTRUCT` *query* $Q_N =$ (`CONSTRUCT` P P)*.*

We use BGP to have a flexibility to represent complex no-values which need more than one triple patterns. We then define an incomplete data source to model the OWA of RDF graphs. As in [4], an *incomplete data source* $\mathcal{G} = (G^a, G^i)$ is a pair of an available graph G^a and an ideal graph G^i such that $G^a \subseteq G^i$. Here, an available graph is the graph that we have, whereas an ideal graph is a possible extension over the available graph, which represents a version of ideal, complete information.

Having no-value statements restricts the possibilities of ideal graphs since they must not contain any instantiations of the information denoted by the statements. Over a graph G, we define the *transfer operator* $T_{\mathcal{N}}(G) = \bigcup_{N \in \mathcal{N}} [\![Q_N]\!]_G$. We define the semantics of no-value statements as follows.

Definition 2 (Satisfaction of No-Value Statements). *An incomplete data source* $\mathcal{G} = (G^a, G^i)$ *satisfies a set* $\mathcal{N}$ *of no-value statements, written as* $\mathcal{G} \models \mathcal{N}$, *if and only if* $T_{\mathcal{N}}(G^i) = \emptyset$.

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

Definition 3 (Query Emptiness). *Let* $\mathcal{G} = (G^a, G^i)$ *be an incomplete data source and* Q *a query. To express that* Q is empty, *we write* $\mathit{Empty}(Q)$. *It is the case that* $\mathcal{G} \models \mathit{Empty}(Q)$ *if and only if* $[\![Q]\!]_{G^i} = \emptyset$.

Query emptiness over one incomplete data source does not always mean that it always holds also over other incomplete data sources. For this reason, we define that the *entailment of a set* $\mathcal{N}$ *of no-value statements and query emptiness* $\mathit{Empty}(Q)$ holds, written as $\mathcal{N} \models \mathit{Empty}(Q)$, if for any incomplete data source $\mathcal{G} \models \mathcal{N}$, we have that $\mathcal{G} \models \mathit{Empty}(Q)$. If the entailment holds, we can guarantee that the query will always return an empty answer no matter which possible extensions of a graph are considered. We have the following theorem to check if a set of no-value statements can guarantee the emptiness of queries.

Theorem 1 (Query Emptiness Entailment from No-Value Statements). *Let* $\mathcal{N}$ *be a set of no-value statements,* Q *be a query, and* $\tilde{P}$ *be the prototypical graph of* Q. *It is the case that* $\mathcal{N} \models \mathit{Empty}(Q)$ *if and only if* $T_{\mathcal{N}}(\tilde{P}) \neq \emptyset$.

Example 1. Let $N = \mathit{No}(\{\,(\mathit{obama}, \mathit{child}, \mathit{?c}), (\mathit{?c}, \mathit{gender}, \mathit{male})\,\})$ be a no-value statement about Obama having no sons. Consider the query $Q = (\{?c, ?s\}, \{\,(\mathit{obama}, \mathit{child}, \mathit{?c}), (\mathit{?c}, \mathit{gender}, \mathit{male}), (\mathit{?c}, \mathit{school}, \mathit{?s})\,\})$ asking for the schools of Obama's sons. We have that $T_{\{N\}}(\tilde{P}) \neq \emptyset$. Thus, from Theorem 1, it holds that $\{N\} \models \mathit{Empty}(Q)$. This means that Q returns an empty answer because of non-existence of the information that is asked, not by the incompleteness of the data source. In contrast, suppose the constant `male` in the query Q were a variable `?g`. If Q returns an empty answer over the data source, that may be due to the incompleteness of the data source.

As seen in the above example, if there is some part of the query that cannot return any answer due to no-value information, then the whole query does not return any answer. Now, we can distinguish between empty query answers from possibly incomplete information, and empty query answers from non-existent information.

3 RDF Representation of No-Value Statements

To concretely represent no-value statements in RDF, we use the reification technique. Given a no-value statement $No(\{\, (s_1, p_1, o_1), \ldots, (s_n, p_n, o_n) \,\})$, we represent the statement as a resource of the class `NoValStatement`, while for each triple pattern, we use a blank node with the properties `subject`, `predicate`, and `object`. Variables are also represented via blank nodes with the property `varName`. Each of the patterns' blank nodes is linked to the statement's resource via the property `hasPattern`. For instance, we represent the no-value statement "Obama has no sons" as follows:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex:   <http://example.org/> .

ex:stmtObama a ex:NoValStatement ; rdfs:label "Obama has no sons" ;
  ex:hasPattern [ ex:subject ex:Obama ; ex:predicate ex:child ;
                  ex:object [ex:varName "c"]] ;
  ex:hasPattern [ ex:subject [ex:varName "c"] ; ex:predicate ex:gender ;
                  ex:object ex:male ] .
```

4 Discussion

In this paper, we present a technique for representing no-value information in RDF and checking the emptiness of queries based on such information. The no-value information restricts possible extensions of an RDF graph wrt. OWA. As a consequence, queries always return an empty answer if they try to capture no-value information. No-value information can also be seen as a stronger version of completeness information, since it also enforces that all possible extensions must contain no corresponding information. Furthermore, when a query is ensured to always return an empty answer, it is obvious that the query is also complete. Hence, this work can complement the completeness reasoning framework for RDF data sources as described in [4]. Another use of no-value information is for data cleaning. If we assume that no-value statements are correct, then we can detect dirty data sources by checking if they contain information that has been stated to be non-existent by the statements. For future work, we will study the relation of our approach to OWL and to more expressive queries.

Acknowledgments The research was supported by the projects "MAGIC: Managing Completeness of Data" funded by the Bolzano province and "CANDy: Completeness-Aware Querying and Navigation on the Web of Data" by UniBZ.

References

1. Fredo Erxleben, Michael Günther, Markus Krötzsch, Julian Mendez, and Denny Vrandecic. Introducing Wikidata to the Linked Data Web. In *ISWC*, 2014.
2. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
3. Steve Harris and Andy Seaborne, editors. *SPARQL 1.1 Query Language*. W3C Recommendation, 21 March 2013.
4. Fariz Darari, Werner Nutt, Giuseppe Pirrò, and Simon Razniewski. Completeness Statements about RDF Data Sources and Their Use for Query Answering. In *ISWC*, 2013.