Visualizing How-Provenance Explanations for SPARQL Queries
Luis Galárraga, Daniel Hernández, Anas Katim, Katja Hose

To cite this version:

HAL Id: hal-04386268
https://inria.hal.science/hal-04386268
Submitted on 10 Jan 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License
ABSTRACT

Knowledge graphs (KGs) are vast collections of machine-readable information, usually modeled in RDF and queried with SPARQL. KGs have opened the door to a plethora of applications such as Web search or smart assistants that query and process the knowledge contained in those KGs. An important, but often disregarded, aspect of querying KGs is query provenance: explanations of the data sources and transformations that made a query result possible. In this article, we discuss the principles behind SPARQLprov and its applications to already deployed SPARQL endpoints. We present visualizations of provenance explanations for SPARQL queries.

KEYWORDS

SPARQL, RDF, how-provenance, query provenance

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:


1 INTRODUCTION

Thanks to the continuous advances in Web information extraction (IE) and knowledge base construction, the Web counts nowadays on more machine-readable data than ever. This data is modeled in RDF and structured in large knowledge graphs (KGs) that can often be queried through public interfaces known as SPARQL endpoints. RDF models knowledge as facts \((s, p, o)\), where predicate \(p\) is also a directed edge from subject \(s\) to object \(o\). KGs allow computers to "understand" the real world and find applications in question answering, Web search, smart assistants, among other scenarios.

A central aspect of querying KGs is query provenance. This is metadata that traces the result of a data transformation back to the records required to obtain this result. We call these records sources. For RDF KGs, these sources are triples, i.e., edges. Query provenance is valuable for KG providers as it makes maintenance tasks such as source selection or view maintenance more efficient [2]. For data consumers, query provenance provides explanations for answers, which users can use to decide, e.g., whether or not to trust a particular output. Formalisms for query provenance include lineage, why-provenance, and how-provenance [4] – the latter being the most expressive. How-provenance structures the sources \(e \in E\) responsible for a query answer into polynomial expressions that form a commutative semiring with natural coefficients \(\mathbb{S} = (\mathbb{N}[E], \otimes, 0, 1)\). Those polynomials define class equivalences and explain how a particular query answer could be derived from the KG. Consider the KG in Figure 1 and the SPARQL query:

\[
\text{SELECT } ?u \text{ WHERE } (?)\text{ occupation ?z. } ?u \text{ nationality :Germany.}
\]

The answer to this query is the binding \(\mu = \{ ?u \rightarrow A.\text{Merkel}\}\), explained by the polynomial \((e_2 \otimes e_4) \otimes e_3\). This tells us that the predicate \(A.\text{Merkel}\) is reported due to the presence of the source triple \(e_3\) and a second source triple that can be either \(e_2\) or \(e_4\). Due to the distributivity of \(\otimes\) over \(\oplus\), the polynomial \((e_2 \oplus e_3) \otimes (e_4 \oplus e_3)\) is also a valid explanation.

Approaches to annotate query answers with how-provenance comprise customized engines [7], engine extensions [6], and methods based on query rewriting layers [1, 5]. The latter approach is particularly appealing for KGs already deployed as SPARQL endpoints. In this demo we illustrate the capabilities of SPARQLprov [5], a query-rewriting method to annotate SPARQL query answers with how-provenance explanations. Unlike state-of-the-art approaches [1, 6, 7], SPARQLprov is natively designed for SPARQL and supports non-monotonic queries, e.g., queries including some notion of set difference. Our demo is a Web application that demonstrates SPARQLprov’s capabilities by visualizing provenance expressions and the subgraphs involved in the generation of answers.

The paper is structured as follows. In Section 2 we elaborate on the background concepts around SPARQLprov. Section 3 guides the user through the functionalities of our demo and elaborates on the technical details behind its implementation. Finally, Section 4 concludes the paper by discussing some perspectives.
Figure 1: An example knowledge graph and a query \( Q \) asking for politicians who are not scientists. The \( e_i \) labels identify the triples stored in the graph.

2 BACKGROUND

2.1 RDF and SPARQL

2.1.1 RDF. KGs are modeled in RDF, a graph data model where entities \( N \) are nodes and labeled edges \( E \) denote relationships between those entities as depicted in Figure 1. Those binary relationships define statements \( t \) structured in triples \((s, p, o)\) such as \((A. Merkel, \text{nationality}, \text{Germany})\). An RDF graph \( G \) is a collection of triples; more formally \( G \subseteq \{(I \cup B) \times I \times T, \text{where} \ I, B, L \text{are sets of IRI}s (webScoped identifiers for real-world concepts), blank nodes (anonymous entities), and literals (strings, numbers, etc.) \}. \( T \subseteq I \cup B \cup L \). KGs are collections of RDF graphs, called RDF datasets, such that each graph is labeled with an IRI except for one of those graphs: the default graph.

2.1.2 SPARQL. We can query RDF datasets using the SPARQL query language. The building blocks of SPARQL queries are triple patterns. A triple pattern \( t \in \{(I \cup B \cup V) \times (I \cup V) \times (E \cup V)\} \) is an RDF triple that allows variables \( ?v \in V \) in its components. Those triple patterns can be grouped into basic graph patterns (BGPs) implicitly combined by the join operator AND. BGPs can, in turn, be combined with different operators such as UNION or MINUS to form graph patterns, such as:

\[
\{?u \text{ occupation } ?e, \ ?u \text{ nationality } \text{Germany}\} \text{ UNION } \{?u \text{ gender male}\}
\]

The evaluation of a graph pattern \( G \) on an RDF graph \( G \) is defined as a function \( [G]\{G\} \) that returns a multiset of mappings \( M \) that match \( G \) according to the SPARQL multiset semantics [3]. If \( G \) is our example graph pattern \( [G]\{G\} = M = \{\{?u \rightarrow A. Merkel, ?z \rightarrow \text{Germany}\}, \{?u \rightarrow E. Macron\}\} \). Graph patterns can be annotated with the graph keyword to restrict the evaluation to a particular RDF graph – otherwise the evaluation is carried out against the default graph. The SPARQL 1.1 specification defines four types of queries on RDF datasets: \textit{SELECT}, \textit{CONSTRUCT}, \textit{DESCRIBE}, and \textit{ASK}. Our demo computes how-provenance explanations for \textit{SELECT} queries, which allow the projection of variables from the mappings returned by the graph evaluation.

2.2 Provenance for RDF and SPARQL

2.2.1 spm-Semirings. How-provenance polynomials are abstract annotations defined on the identifiers of a set of source edges, i.e., triples. These expressions form a commutative semiring \( S = (\mathbb{N}[E], \otimes, \oplus, 0, 1) \). Commutative semirings, however, cannot capture the semantics of non-monotonic queries, i.e., queries with some notion of difference. To cope with this limitation, Geerts et al. [3] propose spm-semirings (spm stands for SPARQL Minus) that extend classical semirings with the \( \ominus \) operator and a set of pertinent axioms that can model SPARQL non-monotonic operators such as \textit{OPTIONAL}, \textit{DIFF} and \textit{MINUS}. The authors also propose the notion of \( S \)-relations, i.e., sets of database mappings \( M \subseteq \text{smp-\text{S}\{-\}-relations} \) that extend classical RDF semirings \( S \subseteq \mathbb{N}[E] \subseteq \mathbb{N}[E] \) with additional \( \sigma \)-relations, i.e., \( S \)-relations derived from our example query in Figure 1 using the spm-seming \( S = (\mathbb{N}[E], \otimes, \oplus, 0, 1) \) with \( E = \{e_1, \ldots, e_7\} \).

2.2.2 Reification. \( \mathbb{N}[E] \)-relations assume that the sources, i.e., KG edges, occurring in the polynomial annotations are identifiable. However, classical RDF triples do not have identifiers, which means that data maintainers must find a mechanism to identify the triples in an RDF graph to provide provenance traces for query answers. This process is called reification and can be implemented in several ways as illustrated in Figure 3. For example, the named graphs reification creates an RDF graph with label \( e_i \) for each triple \( t \) in the original graph. Conversely, the Wikidata reification defines a surrogate entity \( e_i \) that identifies the triple, and two surrogate predicates derived from the original predicate. Those predicates are used to link \( e_i \) to the subject and object of the original triple.

2.3 SPARQLprov

Given a \textit{SELECT} query \( Q \) conceived to run against an RDF graph \( G \), SPARQLprov returns an \( \mathbb{N}[E] \)-relation that annotates each solution mapping in \( [Q]\{G\} \) with a how-provenance polynomial. As previously stated, those polynomials are constructed on top of triple identifiers, hence SPARQLprov assumes that graph \( G \) has been reified into \( G' \). To compute the final \( \mathbb{N}[E] \)-relation, SPARQLprov rewrites the original query \( Q \) into a new query \( Q' \) that is evaluated on \( G' \) according to the standard SPARQL semantics. Hence, the engine returns \( Q \)'s solutions as a standard set of mappings extended with additional \textit{provenance variables} that encode the structure of...
We demonstrate the capabilities of SPARQLprov on the Wikidata public SPARQL endpoint (https://query.wikidata.org/) with our graphical web application, depicted in Figure 5. Wikidata is a popular general knowledge graph that provides reified and non-reified versions of the data.

**Query Editor.** Users can write SPARQL queries in the query editor at the top left of the interface, i.e., section (a) in Fig. 5. In our example we show a query that asks for countries that are not sovereign states. The editor provides a few pre-defined example queries. Query execution is triggered by clicking the button “Run”.

**Queries Panel.** Next to the query editor, i.e., section (b) in Fig. 5, our demo shows two tabs. The Original Query tab shows the query sent to SPARQLprov. This query is identical to the query provided by the user, except that it adds a configurable limit clause designed to avoid timeouts in the SPARQL endpoint. Query rewriting complexifies the actual queries sent to the SPARQL engine, which can lead to timeouts in public endpoints. The Rewritten Query tab displays the results of SPARQLprov’s rewriting.

**Results.** At the bottom right, i.e., Fig. 5 section (c), the demo shows the query solutions annotated with their how-provenance. For queries where the main operation is MINUS, this set also includes the bindings that would have been removed from the actual result set because they match the second operand. Those non-solutions are highlighted in gray, whereas actual solutions are explained by polynomials of the form \( P(\mathcal{U}) \ominus 0 \), where the 0 suggests that they did not match the second operand of \( \text{MINUS} \).
Figure 5: SPARQLprov demo user interface. (a) is the query editor, (b) shows the original and rewritten queries, (c) shows the query answers annotated with how-provenance, (d) is the graph neighborhood of the source triples associated to a solution.

**Neighborhood.** When the user clicks the button next to the provenance of a solution in the results panel, our application shows a subgraph with the source triples referenced in the provenance polynomial – section (d) in Fig 5. That way, the user can visualize the actual RDF statements that explain the presence (and in specific cases the absence) of the solution in the results set. These are highlighted in light blue. The graph view also includes additional “neighbor” edges, which we generate via a `CONSTRUCT` query.

4 CONCLUSION

We have presented a visual application that demonstrates the process of annotating query solutions with how-provenance explanations using the SPARQLprov approach. Thanks to its architecture based on query rewriting, SPARQLprov can be used on top of already deployed engines, and as illustrated in this demo, provide live visual explanations for the results of queries in public SPARQL endpoints. Those explanations hold great potential for visual data exploration in different settings. With this in mind, we envision to develop and study novel approaches for visual provenance in KGs applied to specific tasks such as source verification and query debugging. Our demo is publicly available at http://sparqlprov.cs.aau.dk, and its source code and documentation can be found at https://gitlab.inria.fr/akatim/sparqlprov-demo.

REFERENCES