Visualizing How-Provenance Explanations for SPARQL Queries

Luis Galárraga
Inria
France
luis.galarraga@inria.fr

Daniel Hernández
University of Stuttgart
Germany
daniel.hernandez@ipvs.uni-stuttgart.de

Anas Katim
INSA Rouen
France
anas.katim@insa-rouen.fr

Katja Hose
Aalborg University
Denmark
khose@cs.aau.dk

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 demonstrate, through a Web application, the capabilities of SPARQLprov, an engine-agnostic method that annotates query results with how-provenance annotations. To this end, SPARQLprov resorts to query rewriting techniques, which make it applicable to already deployed SPARQL endpoints. We describe the principles behind SPARQLprov and discuss perspectives on visualizing how-provenance explanations for SPARQL queries.

KEYWORDS

SPARQL, RDF, how-provenance, query provenance

ACM Reference Format:
Luis Galárraga, Daniel Hernández, Anas Katim, and Katja Hose. 2023. Visualizing How-Provenance Explanations for SPARQL Queries. In *Proceedings of The Web Conference (WWW).* ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

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[1] and structured in large knowledge graphs (KGs) that can often be queried through public interfaces known as SPARQL endpoints[2]. RDF models knowledge as facts $\langle s, p, o \rangle$, 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 $e \in \mathcal{E}$ responsible for a query answer into polynomial expressions that form a commutative semiring with natural coefficients $\mathbb{S} = (\mathbb{N}[\mathcal{E}], \otimes, \oplus, 0, 1)$. 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:

```
SELECT ?u WHERE {?u :occupation ?z . ?u :nationality :Germany.}
```

The answer to this query is the binding $\mu = \{?u \rightarrow A.\ Merkel\}$, explained by the polynomial $(e_2 \oplus e_4) \otimes e_3$. This tells us that *A. Merkel* is reported due to the simultaneous 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 $(e_2 \oplus e_3) \oplus (e_4 \otimes e_3)$ 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.

[1] https://www.w3.org/RDF/
[2] https://www.w3.org/TR/2013/REC-sparql11-overview-20130321/

WWW, April 30–May 04, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

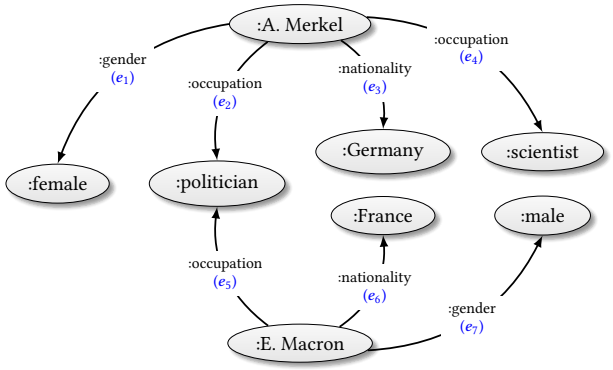

Figure 1: An example knowledge graph and a query Q asking for politicians who are not scientists. The e_i labels identify the triples stored in the graph.

2 BACKGROUND

2.1 RDF and SPARQL

2.1.1 RDF. KGs are modeled in RDF, a graph data model where entities $\mathcal{N}$ are nodes and labeled edges $\mathcal{E}$ denote relationships between those entities as depicted in Figure 1. Those binary relationships define statements t structured in triples $\langle s, p, o \rangle$ such as $\langle$*:A. Merkel, :nationality, :Germany* $\rangle$. An *RDF graph* $\mathcal{G}$ is a collection of triples; more formally $\mathcal{G} \subset (\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times \mathcal{T}$, where $\mathcal{I}$, $\mathcal{B}$, $\mathcal{L}$ are sets of IRIs (web-scoped identifiers for real-world concepts), blank nodes (anonymous entities), and literals (strings, numbers, etc.), and $\mathcal{T} = \mathcal{I} \cup \mathcal{B} \cup \mathcal{L}$. 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* $\hat{t} \in (\mathcal{I} \cup \mathcal{B} \cup \mathcal{V}) \times (\mathcal{I} \cup \mathcal{V}) \times (\mathcal{E} \cup \mathcal{V})$ is an RDF triple that allows variables $?v \in \mathcal{V}$ 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 occupation ?z* . ?*u nationality Germany*} UNION {?*u gender male*}

The evaluation of a graph pattern G on an RDF graph $\mathcal{G}$ is defined as a function $[\![G]\!]_{\mathcal{G}}$ that returns a multiset of mappings $\mathcal{M}$ that match $\mathcal{G}$ according to the SPARQL multiset semantics [3]. If $\mathcal{G}$ is our example graph pattern $[\![G]\!]_{\mathcal{G}} = \mathcal{M} = \{\{?u \rightarrow$ *:A. Merkel*, $?z \rightarrow$ *: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: SELECT, CONSTRUCT, DESCRIBE, and ASK. Our demo computes how-provenance explanations for SELECT queries, which allow the projection of variables from the mappings returned by the graph evaluation.

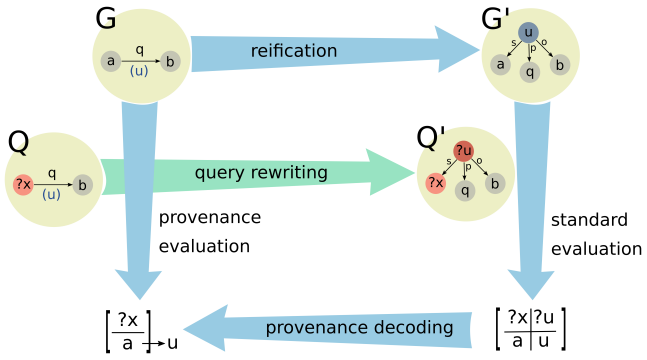

Figure 2: Schema of SPARQLprov's architecture.

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 $\mathbb{S} = (\mathbb{N}[\mathcal{E}], \otimes, \oplus, 0, 1)$. 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 OPTIONAL, DIFF and MINUS. The authors also propose the notion of $\mathbb{S}$-relations, i.e., sets of database mappings $\mu \in \mathcal{M}$ annotated with elements from a semiring $\mathbb{S}$. For example, the relation $\{\{?u \rightarrow$ *A. Merkel*$\} \rightarrow (e_2 \ominus e_3)\}$ is an $\mathbb{N}[\mathcal{E}]$-relation derived from our example query in Figure 1 using the spm-semiring $\mathbb{S} = (\mathbb{N}[\mathcal{E}], \otimes, \oplus, \ominus, 0, 1)$ with $\mathcal{E} = \{e_1, \ldots, e_7\}$.

2.2.2 Reification. $\mathbb{N}[\mathcal{E}]$-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_t for each triple t in the original graph. Conversely, the Wikidata reification defines a surrogate entity $e_t \in \mathcal{I}$ that identifies the triple, and two surrogate predicates derived from the original predicate. Those predicates are used to link e_t to the subject and object of the original triple.

2.3 SPARQLprov

Given a SELECT query Q conceived to run against an RDF graph $\mathcal{G}$, SPARQLprov returns an $\mathbb{N}[\mathcal{E}]$-relation that annotates each solution mapping in $[\![Q]\!]_{\mathcal{G}}$ with a how-provenance polynomial. As previously stated, those polynomials are constructed on top of triple identifiers, hence SPARQLprov assumes that graph $\mathcal{G}$ has been reified into $\mathcal{G}$'. To compute the final $\mathbb{N}[\mathcal{E}]$-relation, SPARQLprov rewrites the original query Q into a new query Q' that is evaluated on $\mathcal{G}$' according to the standard SPARQL semantics. Hence, the engine returns Q's solutions as a standard set of mappings extended with additional *provenance variables* that encode the structure of

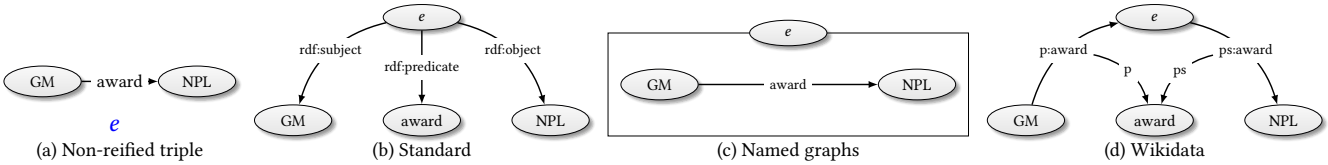

Figure 3: Illustration of different reification schemes for RDF.

?x	?⊖	?⊖1Σ	?⊖1Σ⊙	?⊖2Σ	?⊖2Σ⊙
A. Merkel	$B(AM)$	$B(AM)$	e_1	–	–
E. Macron	$B(EM)$	$B(EM)$	e_2	$B(EM)$	e_3

=

?x	?provenance
A. Merkel	$u_1 \ominus 0$
E. Macron	$u_2 \ominus u_3$

Figure 4: Result of annotating the results of the query from Figure 1 using SPARQLprov. The SPARQL engine returns the table on the left, which is decoded into the table on the right.

the provenance annotations. A decoding phase translates those extended mappings into an $\mathbb{N}[\mathcal{E}]$-relation. The whole process is depicted in Figure 2.

We illustrate SPARQLprov with the example query in Figure 1. The approach rewrites the query into

```
SELECT ?x ?⊖ ?⊖1Σ ?⊖1Σ⊙ ?⊖2Σ ?⊖2Σ⊙ WHERE {
  {Reify(t̂_pol, ?⊖1Σ⊙)} OPTIONAL {Reify(t̂_sci, ?⊖2Σ⊙)}
  BIND (B(?x) AS ?⊖1Σ) BIND (B(?x) AS ?⊖2Σ) BIND (B(?x) AS ?⊖)
},
```

where the function `Reify` rewrites a SPARQL pattern according to the reification scheme used in the target graph $\mathcal{G}$'. For instance, if $\mathcal{G}$' uses the Wikidata reification scheme – illustrated in Figure 3 (d) – `Reify`$(\hat{t}_{pol}, ?\ominus 1\Sigma\odot)$ becomes

:politician p:occupation ?⊖1Σ⊙ . ?⊖1Σ⊙ *ps:occupation :politician*

i.e., the variable ?⊖1Σ⊙ binds to the identifiers that reify the triples matched by the triple pattern $\hat{t}_{pol}$. The results of running this query are shown in Figure 4. Provenance polynomials are tree operators whose structure is encoded in the values and names of the provenance variables. For instance the symbol ⊙ in a variable name tells us that the variable will be bound to actual sources, whereas ⊖1 tells us the source will be the first operand of a `MINUS` operation. The function $B(\cdot)$ receives a set of values as input and generates surrogate IRIs that guide the decoding phase towards the construction of the polynomials. We remark that for non-monotonic queries, such as our example, SPARQLprov may return explanations for bindings that are not actual query solutions. This is the case of *E. Macron* in Figure 4, which matches the first triple pattern but is removed because it matches the second – as explained by $u_2 \ominus u_3$. For details about the rewriting rules as well as the decoding of the variable names into polynomials, we refer the reader to [5].

3 DEMO

In the following we elaborate on our application built to demonstrate SPARQLprov's capabilities. We describe the application's architecture, followed by its interface components explained through a demonstration scenario on the Wikidata SPARQL endpoint.

3.1 Architecture

The demo is implemented as a classical client-server web application. The server, written in Javascript with Nodejs, stands as a middleware in between the client UI, SPARQLprov, and the target SPARQL engine. SPARQLprov is a standalone application with two modules. The *rewriting module* takes a standard SPARQL query as input and rewrites it into a new query that, when executed on the target engine, returns query solutions with provenance annotations. These annotations are encoded as classical relations as illustrated in Figure 4. These relations are converted into query solutions annotated with how-provenance polynomials through the SPARQLprov's *decoder module*. The server orchestrates the communication between the different components, i.e., it rewrites the queries written by the user using SPARQLprov, sends the rewritten queries to the SPARQL endpoint, and invokes the SPARQLprov decoder to compute the final annotations, which are sent back to the client. The client of our Web application is a lightweight user interface written in HTML, CSS, and the Javascript library D3.

3.2 Demonstration Scenario

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 result 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 `MINUS`.

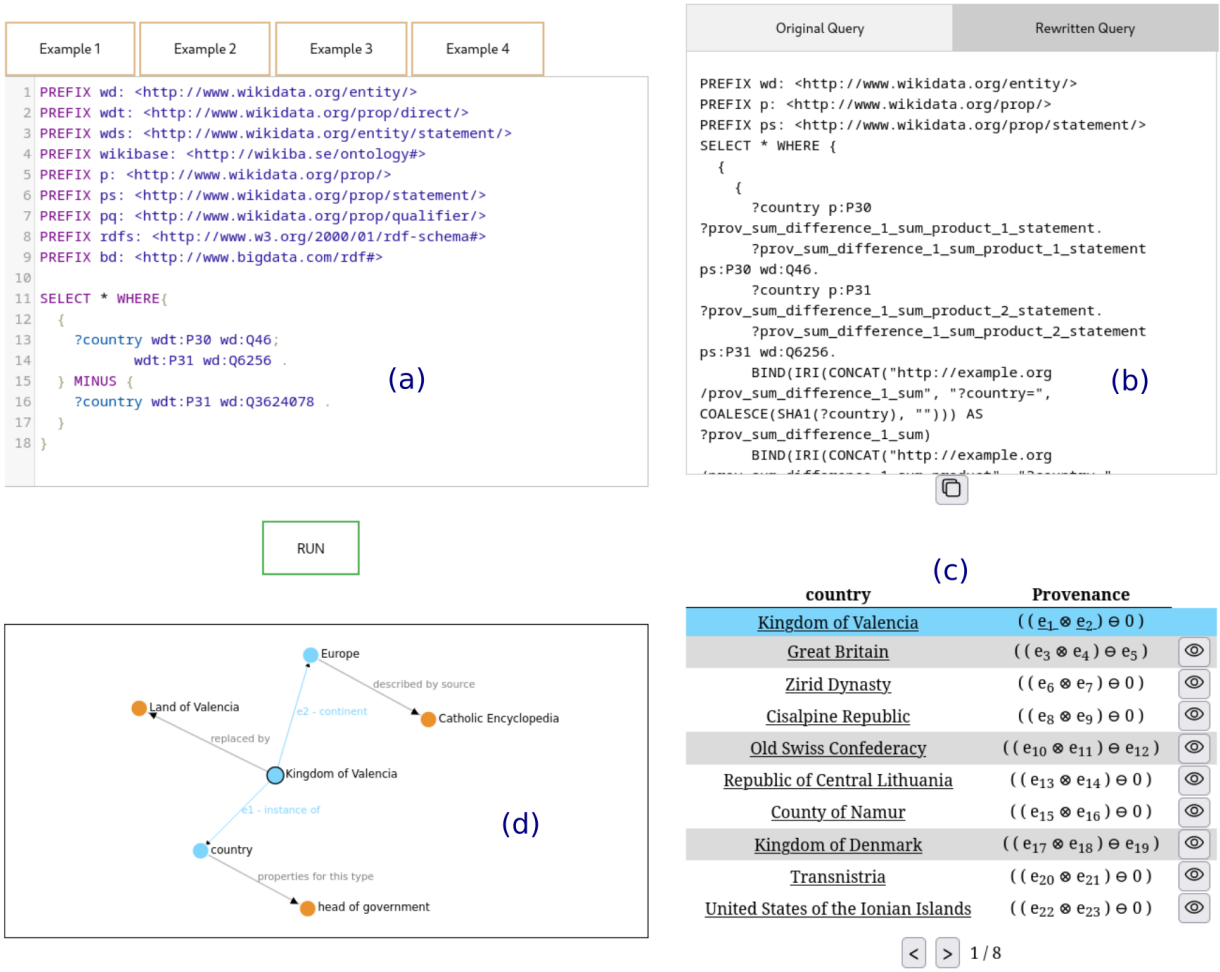

Figure 5: SPARQLprov demo' user interface. (a) is the query editor, (b) shows the original and rewritten queries, (c) shows the query answers annotated with how-provenance, (d) is the graph neighborhood of the source triples associated to a solution.

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/sparlqprov-demo.

REFERENCES

[1] B. S. Arab, Su Feng, B. Glavic, S. Lee, X. Niu, and Q. Zeng. 2018. GProM - A Swiss Army Knife for Your Provenance Needs. *IEEE Data Eng. Bull.* 41, 1 (2018), 51–62.
[2] G. Gaur, A. Bhattacharya, and S. Bedathur. 2020. How and Why is An Answer (Still) Correct? Maintaining Provenance in Dynamic Knowledge Graphs. In *CIKM*.
[3] F. Geerts, T. Unger, G. Karvounarakis, I. Fundulaki, and V. Christophides. 2016. Algebraic Structures for Capturing the Provenance of SPARQL Queries. *J. ACM* 63, 1 (2016), 7:1–7:63.
[4] Boris Glavic. 2021. *Data Provenance: Origins, Apps, Algorithms and Models.* Now Foundations & Trends.
[5] Daniel Hernández, Luis Galárraga, and Katja Hose. 2021. Computing How-Provenance for SPARQL Queries via Query Rewriting. *Proc. VLDB Endow.* 14, 13 (sep 2021), 3389–3401. https://doi.org/10.14778/3484224.3484235
[6] Pierre Senellart, Louis Jachiet, Silviu Maniu, and Yann Ramusat. 2018. ProvSQL: Provenance and Probability Management in PostgreSQL. *PVLDB* 11, 12 (2018), 2034–2037.
[7] M. Wylot, P. Cudré-Mauroux, and P. T. Groth. 2014. TripleProv: Efficient Processing of Lineage Queries in a Native RDF Store. In *WWW*. 455–466.