eSPARQL: Representing and Reconciling Agnostic and Atheistic Beliefs in RDF-star Knowledge Graphs

Xinyi Pan[1(✉)], Daniel Hernández[1], Philipp Seifer[2], Ralf Lämmel[2], and Steffen Staab[1,3]

[1] Institute for Artifical Intelligence, University of Stuttgart, Stuttgart, Germany
xinyi.pan@ms.informatik.uni-stuttgart.de,
{daniel.hernandez,steffen.staab}@ki.uni-stuttgart.de
[2] The Software Languages Team, University of Koblenz, Koblenz, Germany
{pseifer,laemmel}@uni-koblenz.de
[3] Web and Internet Science Research Group, University of Southampton, Southampton, UK

Abstract. Over the past few years, we have seen the emergence of large knowledge graphs combining information from multiple sources. Sometimes, this information is provided in the form of assertions about other assertions, defining contexts where assertions are valid. A recent extension to RDF which admits statements over statements, called RDF-star, is in revision to become a W3C standard. However, there is no proposal for a semantics of these RDF-star statements nor a built-in facility to operate over them. In this paper, we propose a query language for epistemic RDF-star metadata based on a four-valued logic, called eSPARQL. Our proposed query language extends SPARQL-star, the query language for RDF-star, with a new type of `FROM` clause to facilitate operating with multiple and sometimes conflicting beliefs. We show that the proposed query language can express four use case queries, including the following features: (i) querying the belief of an individual, (ii) the aggregating of beliefs, (iii) querying who is conflicting with somebody, and (iv) beliefs about beliefs (i.e., nesting of beliefs).

Keywords: SPARQL · RDF-star · knowledge graphs · epistemic querying

1 Introduction

Over the past few years, we have seen the emergence of large knowledge graphs combining information from multiple sources, such as Wikidata [21], YAGO [20], and DBpedia [3]. These knowledge graphs provide information about a great variety of entities, such as people, countries, universities, as well as facts over these entities, which are codified as triples. For example, the fact that Albert Einstein was born in Germany can be encoded as a triple (Einstein, wasBorn, Germany).

G. Demartini et al. (Eds.): ISWC 2024, LNCS 15232, pp. 155–172, 2025.
https://doi.org/10.1007/978-3-031-77850-6_9

Triples like this are the information units of the Resource Description Framework (RDF) [12], a data model for representing information about World Wide Web resources, and SPARQL [10], the query language proposed by the W3C to query RDF data.

RDF and SPARQL have been built on the simplifying assumption that statements (triples) are either *true* or *unknown*, leading to a representation of knowledge that can never conflict with each other, and thus can never lead to inconsistencies. To present competing statements, knowledge bases include reified statements without introducing them as a stated statement. For example, the statement "Jesus is a deity" is provided in a reified form in Wikidata, and annotated as "supported by Christianity, Messianic Judaism, and Manichaeism" but "disputed by Islam, Atheism, and Judaism". Acknowledging the need to make statements about statements, two recent extensions, namely RDF-star and SPARQL-star [11], were proposed. However, these extensions do not deviate from the assumption that statements are either true or unknown.

To illustrate the need to work with statements that are *true*, *unknown*, *false*, or *conflicted*, let us consider some disagreements on beliefs in the Christian religion. Several councils were convened to establish consensus about different aspects of the nature of Jesus, his birth, his mother's birth, his father, and his existence before his birth. By stating a dogma, a council established what should be considered true and what is a heresy. For example, one of these beliefs can be encoded in RDF-star as a triple where the last element is also a triple:

$$(\mathsf{PopeDI}, \mathsf{believesToBeTrue}, (\mathsf{Jesus}, \mathsf{a}, \mathsf{FullDeity})).$$

RDF-star does not provide a semantics for this statement, but we can assume that it means "Pope Damasus the First believes that Jesus is a full Deity." Indeed, the Pope presided over the Council of Nice, where this statement became a dogma. This belief was not shared by all participants of the Council. Arius believed that Jesus was created by decision of God, and thus his deity is not as full as the deity of his father. Arius' belief can be encoded as follows:

$$(\mathsf{Arius}, \mathsf{believesToBeFalse}, (\mathsf{Jesus}, \mathsf{a}, \mathsf{FullDeity})).$$

The two statements above do not entail that "Jesus is a full deity" nor the contrary, but ascribe some different people's beliefs. Since we have contradictory opinions between Christians, we can say that this statement is conflicted according to Christianity:

$$(\mathsf{Christianity}, \mathsf{believesToBeConflicted}, (\mathsf{Jesus}, \mathsf{a}, \mathsf{FullDeity})).$$

On the other hand, the mathematician Bertrand Russell described his religious beliefs as follows: *"In regard to the Olympic gods, speaking to a purely philosophical audience, I would say that I am an Agnostic. But speaking popularly, I think that all of us would say in regard to those gods that we were Atheists. In regard to the Christian God, I should, I think, take exactly the same*

line." Regarding the aforementioned statement, Russell's beliefs can be encoded as follows:

$$(\mathsf{Russell}, \mathsf{believesToBeUnknown}, (\mathsf{Jesus}, \mathsf{a}, \mathsf{FullDeity})),$$
$$(\mathsf{Russell}, \mathsf{believesToBeUnknown}, (\mathsf{Zeus}, \mathsf{a}, \mathsf{FullDeity})).$$

Russell defines himself as agnostic, since he cannot decide whether the statement is false or true, and he also describes his beliefs about the beliefs of Christians who may be believers regarding Jesus, but atheists regarding an Olympic deity. This Russell's argument applied to PopeDI is encoded as follows:

$$(\mathsf{Russell}, \mathsf{believesToBeTrue}, (\mathsf{PopeDI}, \mathsf{believesToBeTrue}, (\mathsf{Jesus}, \mathsf{a}, \mathsf{FullDeity}))),$$
$$(\mathsf{Russell}, \mathsf{believesToBeTrue}, (\mathsf{PopeDI}, \mathsf{believesToBeFalse}, (\mathsf{Zeus}, \mathsf{a}, \mathsf{FullDeity}))).$$

In order to deal with multiple views of the world, it becomes necessary to operate on sets of facts. For example, we would like to execute a SPARQL query against the set of all facts that are valid according to Christianity, or a given person. However, SPARQL does hardly support working with sets of facts. Indeed, SPARQL has an ambivalent attitude towards sets of facts, as it allows constructing such using `FROM`, but its main operators all work on graph pattern matching. Aside from the SPARQL `UPDATE` functionality, the only operators to do so are `FROM` and `FROM NAMED`. We propose to extend the `FROM` functionality of SPARQL and call this extension eSPARQL to establish actual sets of facts based on which triple pattern matching and downstream operations (`FILTER`, `AGGREGATE`, etc.) may be applied.

eSPARQL allows constructing graphs in `FROM` clauses filtering triples with patterns that can include epistemic conditions. For example, the following eSPARQL query asks for all full deities according to Christianity.

```
SELECT ?deity
FROM BELIEF <Christianity>
WHERE { ?deity a <FullDeity> }
```

Intuitively, the clause `FROM BELIEF <Christianity>` generates a graph whose triples are marked as either *true*, *false*, *unknown*, or *conflicting*. These truth values represent the `<Christianity>` beliefs. The graph that results from extracting the Pope beliefs is then used for matching the triple pattern `?deity a <FullDeity>`, and returning the bindings for variable `?deity`. The answers for Jesus, his father God, and his mother Maria will be annotated as conflicted, true, and false, respectively, and individuals that are not mentioned will be annotated as unknown.

This paper makes the following contributions:

1. We describe use case requirements for an epistemic query language (Sect. 2).
2. We provide a semantics for RDF-star and SPARQL-star as a $\mathcal{K}$-annotated algebra, which operates over functions from SPARQL-star solution mappings to $\mathcal{K}$ elements (Sect. 4). This algebra extends the Geerts et al. $\mathcal{K}$-annotated algebra [6], which is used as the foundation for how-provenance in SPARQL.

3. We propose a query language, eSPARQL, an extension to the $\mathcal{K}$-annotated SPARQL-star where the abstract semiring structure $\mathcal{K}$ is substituted by the concrete semirings in the $\mathcal{FOUR}$ structure (Sect. 5).
4. We show that eSPARQL can express the use case queries, and present an user interface for the eSPARQL algebra (Sect. 6).
5. We show that SPARQL queries can lead to query results that cannot be encoded with finite expressions using the support of the solution (Sect. 7). We propose an approach to deal with these infinite results.

2 Use Case Requirements

We have explored a broad range of queries that an epistemic query language for RDF-star should be able to answer. We distil four query examples that this language must allow formulating easily. They require (i) to query for people's beliefs, (ii) to query for aggregated or integrated belief states, (iii) to query for subjects whose beliefs coincide, conflict, are compatible or ignorant of other belief sets, (iv) allow for nested belief states.

Use Case U1 (Query for People's Beliefs). We should be able to query people's beliefs regarding a given statement, and the query should return the bindings to the possible variables in the statement and the respective four-valued logic truth values for the variable bindings. The expression for these queries should be simple (i.e., do not include any algebraic operators like AND, UNION, FILTER, or SELECT).

Example: Return the Full Deities `?x` *According to Pope Damasus the First.* All returned individuals should be annotated with a belief value that indicates the belief of the Pope regarding the individual. For example, Jesus must be annotated as *true* if the Pope believes Jesus is a full deity, *false* if the Pope believes Jesus is not a full deity, *unknown* if there is no information about the belief of the Pope regarding the nature of Jesus, and *conflicted* if either the Pope believes this statement is conflicted, or the belief is both *true* and *false*.

Use Case U2 (Query for Combined Belief Values). We should be able to combine people's beliefs to represent the belief of a group, or operations between different people's beliefs.

Example: Return the Beliefs of Christian People Regarding the Full Deities. For example, Jesus must be annotated with true if all Christians think Jesus is a *full* deity, *false* if all Christians think Jesus is not a full deity, *conflicted* if they have contradictory opinions, and unknown if no Christian has an opinion regarding the nature of Jesus.

Use Case U3 (Query for People Whose Beliefs Coincide, Conflict, or are Compatible). Given a set of statements, we should be able to query people who coincide (they all have the same belief for all the statement), conflict (some people believe a statement is true, whereas others believe it is false), or they are

compatible (if a person believes a fact is true, the others can believe it is true or unknown).

Example: Return All People Whose Opinion is Conflicted with Pope Damasus the First in at Least One Statement. For example, since Pope Damasus the First believes that Jesus is a full deity, and Arius believes it is not, then Arius should be returned (i.e., annotated with true). Since Russell is agnostic regarding an opinion of Pope Damasus the First, Russell is not in conflict with the Pope, so Russell must be annotated with false.

Use Case U4 (Query Nested Belief States). We should be able to query about what people believe other people believe.

Example: Return all the People x such that there Exists a Person y that Believes that x Believes that Zeus is Not a Full Deity. In our example in the introduction, we must annotate Pope Damasus the First with true, because Russell believes the pope believes Zeus is not a full deity. Other individuals should be annotated with unknown.

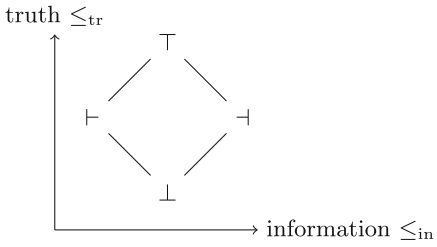
3 Preliminaries

The $\mathcal{FOUR}$ Structure. A *partially ordered set* (or *poset*) is a set taken together with a partial order on it. Formally, a partially ordered set is defined as a pair $(A, \leq)$, where A is called the *ground set* of the poset and $\leq$ is the *partial order* of the poset.

A poset $(A, \leq)$ defines a lattice if for every two elements $a, b \in A$, there is a unique maximum element $c \in A$ such that $c \leq a$ and $c \leq b$, called the *meet* of a and b, and there is a unique minimum element $c \in A$ such that $a \leq c$ and $b \leq c$, called the *join* of a and b.

We write $\mathcal{FOUR}$ to denote the set $\{\bot, \top, \vdash, \dashv\}$, and we use $\mathcal{FOUR}$ *elements* to refer to the elements in this set. We define the posets $(\mathcal{FOUR}, \leq_{\text{tr}})$ and $(\mathcal{FOUR}, \leq_{\text{in}})$ as the minimum posets satisfying the inequalities $\bot \leq_{\text{tr}} \vdash$, $\bot \leq_{\text{tr}} \dashv$, $\vdash \leq_{\text{tr}} \top$, $\dashv \leq_{\text{tr}} \top$, $\vdash \leq_{\text{in}} \bot$, $\vdash \leq_{\text{in}} \top$, $\bot \leq_{\text{in}} \dashv$, and $\top \leq_{\text{in}} \dashv$. These two posets with ground set $\mathcal{FOUR}$, define the bilattice depicted in Fig. 1. The axis can be interpreted as the orderings regarding truth and information, and the $\mathcal{FOUR}$ elements are the respective elements of the 4-valued logic *false* ($\bot$), *true* ($\top$), *unknown* ($\vdash$), and *conflicting* ($\dashv$). We use $\mathcal{FOUR}$ *operators* to refer to the operators $\oslash$, $\otimes$, $\ominus$, and $\oplus$ (see Fig. 1), and $\mathcal{FOUR}$ *structure* to refer to the algebraic structure that consists of the $\mathcal{FOUR}$ operators over the $\mathcal{FOUR}$ set. Given a $\mathcal{FOUR}$ operator $\circ$ and a set $A = \{\alpha_1, \ldots, \alpha_n\} \subseteq \mathcal{FOUR}$, we write $\text{Id}(\circ)$ to denote the identity element for an operator $\circ$ in the $\mathcal{FOUR}$ structure (i.e., $\text{Id}(\oslash) = \top$, $\text{Id}(\otimes) = \bot$, $\text{Id}(\ominus) = \dashv$, and $\text{Id}(\oplus) = \vdash$), we write $\text{Abb}(\circ)$ to denote the absorbing element for a $\mathcal{FOUR}$ operator $\circ$, (i.e., $\text{Abb}(\oslash) = \bot$, $\text{Abb}(\otimes) = \top$, $\text{Abb}(\ominus) = \vdash$, and $\text{Abb}(\oplus) = \dashv$), and we write $\circ(A)$ for the $\mathcal{FOUR}$ element $\text{Id}(\circ) \circ \alpha_1 \circ \cdots \circ \alpha_n$.

A *monoid* $\mathcal{M}$ is an algebraic structure $(M, +_{\mathcal{M}}, 0_{\mathcal{M}})$ where M is a non-empty set, $+_{\mathcal{M}}$ is a closed associative operation on set M, and $0_{\mathcal{M}} \in M$ is the



$\mathcal{FOUR}$ operators:

- $\alpha \otimes \beta$ and $\alpha \oplus \beta$ are the respective meet and join of α and β in poset $(\mathcal{FOUR}, \leq_{\mathrm{tr}})$.
- $\alpha \ominus \beta$ and $\alpha \oslash \beta$ are the respective meet and join of α and β in poset $(\mathcal{FOUR}, \leq_{\mathrm{in}})$.

Fig. 1. Bilattice of the $\mathcal{FOUR}$-structure.

identity element of operation $+_{\mathcal{M}}$ (i.e., $0_{\mathcal{M}} +_{\mathcal{M}} a = a$ and $a +_{\mathcal{M}} 0_{\mathcal{M}} = a$, for every $a \in M$). The monoid $\mathcal{M}$ is said to be *commutative* if operation $+_{\mathcal{M}}$ is commutative.

A *semiring* $\mathcal{K}$ is an algebraic structure $(K, +_{\mathcal{K}}, \times_{\mathcal{K}}, 0_{\mathcal{K}}, 1_{\mathcal{K}})$ where $(K, +_{\mathcal{K}}, 0_{\mathcal{K}})$ and $(K, \times_{\mathcal{K}}, 1_{\mathcal{K}})$ are monoids, and $\times_{\mathcal{K}}$ distributes over $+_{\mathcal{K}}$ (i.e., $a \times_{\mathcal{K}} (b +_{\mathcal{K}} c) = (a \times_{\mathcal{K}} b) +_{\mathcal{K}} (a \times_{\mathcal{K}} c)$ and $(b +_{\mathcal{K}} c) \times_{\mathcal{K}} a = (b \times_{\mathcal{K}} a) +_{\mathcal{K}} (c \times_{\mathcal{K}} a)$). The semiring $\mathcal{K}$ is said to be *commutative* if operations $+_{\mathcal{K}}$ and $\times_{\mathcal{K}}$ are commutative. For example, the algebraic structures $(\mathcal{FOUR}, \otimes, \oplus, \bot, \top)$, $(\mathcal{FOUR}, \ominus, \oslash, \vdash, \dashv)$, $(\{\bot, \top\}, \otimes, \oplus, \bot, \top)$, and $(\mathbb{N}, +, \cdot, 0, 1)$ are commutative semirings.

We will write $\mathcal{FOUR}_{\mathrm{th}}$ and $\mathcal{FOUR}_{\mathrm{in}}$ for the semirings $(\mathcal{FOUR}, \otimes, \oplus, \bot, \top)$ and $(\mathcal{FOUR}, \ominus, \oslash, \vdash, \dashv)$, respectively.

4 $\mathcal{K}$-annotated SPARQL-star

In this section, we extend a fragment of the Geerts et al. [6] $\mathcal{K}$-annotated SPARQL algebra to support RDF-star and SPARQL-star (following the notions from the working draft [11]). We consider a fragment without the non-monotonic operator `MINUS` because the non-monotonic operators go beyond the semiring structure of each of the two $\mathcal{FOUR}$ lattices.

We assume two countable pairwise disjoint sets $\mathbf{I}$ and $\mathbf{V}$, called the set of *IRIs* and the set of *variables*. We call the elements of $\mathbf{I}^3$ *RDF triples*[1]. Given an RDF triple $(s, p, o) \in \mathbf{I}^3$, we say that s is the *subject*, p is the *predicate*, and o is the *object* of the triple. According to the working draft [11], *SPARQL-star triples* and *SPARQL-star triple patterns* are defined recursively as follows. Given two SPARQL-star triple patterns T_1 and T_2, and an RDF triple pattern (S, P, O), the triples (S, P, O), (T_1, P, O), (S, P, T_2), and (T_1, P, T_2) are SPARQL-star triple patterns. A SPARQL-star triple pattern without variables is an *RDF-star triple* (and $\mathcal{T}$ is the set of all RDF-star triples), and an *RDF-star graph* is a set of RDF-star triples.

A *SPARQL-star solution mapping* (or a *mapping*) is a partial function $\mu : \mathbf{V} \to \mathbf{I} \cup \mathcal{T}$ with finite domain $\mathrm{dom}(\mu)$. Two mappings μ and μ' are *compatible*

[1] In RDF, triples can also have other element types, namely blank nodes and literals. For simplicity, we do not consider them, since they do not change our results.

if $\mu(\texttt{?x}) = \mu'(\texttt{?x})$ for every variable $\texttt{?x} \in \text{dom}(\mu) \cap \text{dom}(\mu')$. If mappings μ and μ' are compatible, $\mu \cup \mu'$ is the mapping with domain $\text{dom}(\mu) \cup \text{dom}(\mu')$ that is compatible with μ and μ'. Given a set of variables $W \subset \mathbf{V}$, we write $\Omega|_W$ to denote the set of mappings $\{\mu \ : \ \text{dom}(\mu) = W\}$.

We next extend RDF-star and SPARQL-star to support annotations over a semiring $\mathcal{K}$. Given a commutative semiring $\mathcal{K} = (K, +_{\mathcal{K}}, \times_{\mathcal{K}}, 0, 1)$, an *RDF-star $\mathcal{K}$-graph* is a function $G : \mathcal{T} \to K$ that maps every RDF-star triple to an element of the semiring $\mathcal{K}$.

Remark 1. Notice that $\mathcal{K}$ denotes an arbitrary semiring, whereas $\mathcal{FOUR}_{\text{th}}$ and $\mathcal{FOUR}_{\text{in}}$ are two concrete semirings. Thus, the $\mathcal{K}$-annotated SPARQL algebra by Geerts et al. [6] is defined in abstract, whereas the $\mathcal{FOUR}_{\text{th}}$-annotated and the $\mathcal{FOUR}_{\text{in}}$-annotated SPARQL-star algebras are concrete instances of the abstract algebra proposed by Geerts et al.

Definition 1 (Syntax of $\mathcal{K}$-annotated SPARQL-star). *We next present a recursive definition for the syntax of* $\mathcal{K}$-annotated SPARQL-star *queries Q and a finite set of variables, called the* in-scope variables*, denoted* $\text{inScope}(Q)$*.*

1. *A SPARQL-star triple pattern T is a $\mathcal{K}$-annotated SPARQL-star query, whose in-scope variables are the variables occurring in triple pattern T.*
2. *Given a $\mathcal{K}$-annotated SPARQL-star query Q and a set of variables $W \subseteq \text{inScope}(Q)$, the expression $(\textsf{SELECT}\, W\, Q)$ is a $\mathcal{K}$-annotated SPARQL-star query with in-scope variables W.*
3. *Given a $\mathcal{K}$-annotated SPARQL-star query Q and a formula φ, the expression $(Q\ \textsf{FILTER}\ \varphi)$ is a $\mathcal{K}$-annotated SPARQL-star query with in-scope variables $\text{inScope}(Q)$.*
4. *Given two $\mathcal{K}$-annotated SPARQL-star queries Q_1 and Q_2, the expression $(Q_1\ \textsf{AND}\ Q_2)$ is a $\mathcal{K}$-annotated SPARQL-star query with in-scope variables $\text{inScope}(Q_1) \cup \text{inScope}(Q_2)$.*
5. *Given two $\mathcal{K}$-annotated SPARQL-star queries Q_1 and Q_2 with the same set W of in-scope variables, the expression $(Q_1\ \textsf{UNION}\ Q_2)$ is a $\mathcal{K}$-annotated SPARQL-star query with in-scope variables W.*

Definition 2 (Semantics of $\mathcal{K}$-annotated SPARQL-star). *Given a semiring $\mathcal{K} = (K, +_{\mathcal{K}}, \times_{\mathcal{K}}, 0_{\mathcal{K}}, 1_{\mathcal{K}})$, the result of evaluating a $\mathcal{K}$-annotated SPARQL-star query Q over a $\mathcal{K}$-graph G, is a function $[\![Q]\!]_G : \Omega|_{\text{inScope}(Q)} \to K$, called* $\mathcal{K}$-relation*, defined recursively as follows:*

1. *Given a SPARQL-star triple pattern T, if $\text{dom}(\mu) = \text{inScope}(T)$ then $[\![Q]\!]_G(\mu)$ is $G(t)$, where t is the RDF-star triple resulting from replacing every variable $\texttt{?x}$ occurring in T with $\mu(\texttt{?x})$; otherwise, $[\![Q]\!]_G(\mu) = 0_{\mathcal{K}}$.*
2. *For the recursive cases, the semantics is the following:*
   (a) $[\![(\textsf{SELECT}\, W\, Q)]\!]_G(\mu) = \sum_{\mu' : \mu'|_W = \mu} [\![Q]\!]_G(\mu')$,
   (b) $[\![(Q\ \textsf{FILTER}\ \varphi)]\!]_G(\mu) = [\![Q]\!]_G(\mu) \times_{\mathcal{K}} k_{\mu \models \varphi}$,
   (c) $[\![(Q_1\ \textsf{UNION}\ Q_2)]\!]_G(\mu) = [\![Q_1]\!]_G(\mu) +_{\mathcal{K}} [\![Q_2]\!]_G(\mu)$,
   (d) $[\![(Q_1\ \textsf{AND}\ Q_2)]\!]_G(\mu) = [\![Q_1]\!]_G(\mu|_{\text{inScope}(Q_1)}) \times_{\mathcal{K}} [\![Q_2]\!]_G(\mu|_{\text{inScope}(Q_2)})$,

where $\sum$ denotes sums using $+_{\mathcal{K}}$, $\mu'|_W$ is the projection of mapping μ' on the variables in W, and $k_{\mu \models \varphi}$ is $1_{\mathcal{K}}$ if replacing the variables in φ according to μ leads to a true formula, and $k_{\mu \models \varphi}$ is $0_{\mathcal{K}}$, otherwise.

Table 1. RDF-star triples in the running example graph $G = \{t_1, \ldots, t_8\}$. The RDF-star triples t_9, t_{10}, and t_{11} are not stated in G but occur nested in the triples in G.

Id	Triple
t_1	(PopeDI, believesToBeTrue, (Jesus, a, FullDeity)),
t_2	(Arius, believesToBeFalse, (Jesus, a, FullDeity)),
t_3	(Christianity, believesToBeConflicted, (Jesus, a, FullDeity)),
t_4	(Russell, believesToBeUnknown, (Jesus, a, FullDeity))
t_5	(Russell, believesToBeTrue, (PopeDI, believesToBeTrue, (Jesus, a, FullDeity)))
t_6	(Russell, believesToBeTrue, (PopeDI, believesToBeFalse, (Zeus, a, FullDeity)))
t_7	(PopeDI, a, Christian)
t_8	(Arius, a, Christian)
t_9	(Jesus, a, FullDeity),
t_{10}	(PopeDI, believesToBeFalse, (Zeus, a, FullDeity))
t_{11}	(Zeus, a, FullDeity)

Example 1 (Running Example). Let $G = \{t_1 \mapsto \top, \ldots, t_8 \mapsto \top, * \mapsto \vdash\}$ be the RDF-star $\mathcal{FOUR}_{\text{in}}$-graph such that $G(t_i) = \top$ for every RDF-star triple t_i, with $1 \leq i \leq 8$, listed in Table 1, and $G(t') = \vdash$ for the rest of all possible RDF-star triples t'. Then, $[\![(\texttt{?x}, \mathsf{believesToBeFalse}, (\texttt{?y}, \mathsf{a}, \mathsf{FullDeity}))]\!]_G$ returns the $\mathcal{K}$-relation $R = \{\{\texttt{?x} \mapsto \mathsf{Arius}, \texttt{?y} \mapsto \mathsf{Jesus}\} \mapsto \top, * \mapsto \vdash\}$. Intuitively, this query returns all people who believe it to be false that somebody is a full deity.

Support of Functions. So far, we have described the $\mathcal{FOUR}$ structure, and the extended Geerts et al. [6] $\mathcal{K}$-annotated algebra. In principle, the elements of this algebra, $\mathcal{K}$-graphs and $\mathcal{K}$-relations, can have no finite representation. One of the concepts used to characterize $\mathcal{K}$-graphs and $\mathcal{K}$-relations with a finite representation is the support of a function, which we describe next.

Given two functions $f : A \to B$ and $g : A \to B$, and a binary operation $\diamond$ closed in B, $f \diamond g$ is the function such that $(f \diamond g)(x) = f(x) \diamond g(x)$. If $\diamond$ has an identity element $\mathrm{Id}(\diamond)$, the *support* of f regarding operation $\diamond$ is the set $\mathrm{supp}_{\diamond}(f) = \{a \in A \ : \ f(a) \neq \mathrm{Id}(\diamond)\}$. If the support of f is a finite set $\{a_1, \ldots, a_n\}$, we encode f with the finite mapping $\{a_1 \mapsto f(a_1), \ldots, a_n \mapsto f(a_n), * \mapsto \mathrm{Id}(\diamond)\}$, where $*$ is a wildcard for the (possibly infinite) remaining mappings that are mapped to the identity of the operation.

Remark 2. Intuitively, the support represents all values that are assigned to *non-zero* values. In the case of a semiring $\mathcal{K} = (K, +_{\mathcal{K}}, \times_{\mathcal{K}}, 0_{\mathcal{K}}, 1_{\mathcal{K}})$ the zero value is

$0_{\mathcal{K}}$, whereas in the $\mathcal{FOUR}$ structure we can consider two zero values, namely $\bot$ and $\vdash$, depending on which semiring we are considering. These two values are the corresponding bottoms of the two lattices in the $\mathcal{FOUR}$ structure. Intuitively, the zero values in $\mathcal{FOUR}$ represent the default value for statements that are not included in a knowledge base, according to either the *closed-world assumption* or the *open-world assumption*.

Remark 3. According to the specification, an RDF-star graph is a finite set of RDF-star triples $\{T_1, \ldots, T_n\}$ and under the open-world assumption (which is the standard for RDF-star) a triple T is true if $T \in G$ and unknown if $T \notin G$. Hence, the RDF-star graph G can be interpreted as the $\mathcal{FOUR}_{\text{in}}$-graph that is represented by the finite mapping $\{T_1 \mapsto \top, \ldots, T_n \mapsto \top, * \mapsto \vdash\}$. Furthermore, the evaluation of a $\mathcal{K}$-annotated SPARQL-star query over a $\mathcal{K}$-graph with finite support always returns a $\mathcal{K}$-relation with finite support. This finite property, guarantees that the algebra can be implemented.

Proposition 1. *Given a semiring $\mathcal{K}$, for every RDF-star $\mathcal{K}$-graph G and every SPARQL-star query Q, the $\mathcal{K}$-relation $[\![Q]\!]_G$ has finite support.*

Proof. This can be shown by induction on the structure of the query.

Remark 4. The $\mathcal{FOUR}_{\text{in}}$-annotated SPARQL-star algebra allows us to query $\mathcal{FOUR}_{\text{in}}$-graphs, but does not provide the means to easily formulate the epistemic queries described in Sect. 2. This motivates the proposal of the query language we describe in the next section.

5 Epistemic SPARQL

This section presents the syntax and semantics of Epistemic SPARQL (eSPARQL), the language designed for the use cases described in Sect. 2.

So far, we have defined the notion of SPARQL-star $\mathcal{K}$-graphs, where $\mathcal{K}$ is a semiring. However, a $\mathcal{K}$-graph is an abstract notion, since no concrete semiring is provided. If concrete semirings like $\mathcal{FOUR}_{\text{th}}$ and $\mathcal{FOUR}_{\text{in}}$ are considered, then we have concrete notions as $\mathcal{FOUR}_{\text{th}}$-graphs and $\mathcal{FOUR}_{\text{in}}$-graphs (see Example 1). In what follows, a $\mathcal{FOUR}$-graph will be a function that associates RDF-star triples to elements in set $\mathcal{FOUR}$, without choosing one of the two semirings ($\mathcal{FOUR}_{\text{th}}$ or $\mathcal{FOUR}_{\text{in}}$) defined by the $\mathcal{FOUR}$ structure.

A key characteristic of $\mathcal{FOUR}$-graphs is that they can contain epistemic metadata encoded using four predicates: $\mathsf{believesToBeTrue}$, $\mathsf{believesToBeFalse}$, $\mathsf{believesToBeUnknown}$, and $\mathsf{believesToBeConflicted}$. Each of these predicates encodes a belief. For example, triple $t_1 = (\mathsf{PopeDI}, \mathsf{believesToBeTrue}, t_9)$ from Table 1 encodes that PopeDI believes statement t_9 to be true.

A basic operation over a $\mathcal{FOUR}$-graph G is thus extracting beliefs for each of these predicates. We next present *belief queries*, which are expressions to extract a $\mathcal{FOUR}$-graph G' representing the belief of one or more people according to the information of an input $\mathcal{FOUR}$-graph G. An atomic belief query $[\mathsf{PopeDI}, \top, \vdash]$

represents all statements PopeDI believe to be $\top$, assuming that the rest are annotated with the state $\vdash$. Compound belief queries are generated by combining atomic statements with the $\mathcal{FOUR}$ operators.

Definition 3 (Syntax of a Belief Query). *A* belief query E *is recursively defined as follows:*

1. *Given an element* $u \in \mathbf{I} \cup \mathbf{V}$*, and two* $\mathcal{FOUR}$ *elements* α *and* β*, the triple* $[u, \alpha, \beta]$ *is an* atomic *belief query.*
2. *Given two belief queries* E_1 *and* E_2*, and a* $\mathcal{FOUR}$*-operator* $\circ$*, the expression* $(E_1 \circ E_2)$ *is a* compound *belief query.*

We write $\mathrm{var}(E)$ *to denote the set of variables occurring in a belief query* E*. A belief query with no variables is said to be* ground*. We write* $(u, \circ)$ *to denote the belief query* $[u, \top, Id(\circ)] \circ [u, \bot, Id(\circ)] \circ [u, \vdash, Id(\circ)] \circ [u, \dashv, Id(\circ)]$.

Definition 4 (Semantics of a Ground Belief Query). *Let* $\circ$ *be a* $\mathcal{FOUR}$*-operator. The semantics of a belief query* E *over a* $\mathcal{FOUR}$*-graph* G *is given by the* $\mathcal{FOUR}$*-graph, denoted* $G|_E$*, defined recursively as follows:*

1. *Given an element* $a \in \mathbf{I}$ *and a* $\mathcal{FOUR}$ *operation* $\circ$*,*

$$G|_{[a,\top,\beta]}(t) = \begin{cases} \top & \text{if } G((a, \mathsf{believesToBeTrue}, t)) \in \{\top, \dashv\}, \\ \beta & \text{otherwise.} \end{cases}$$

$$G|_{[a,\bot,\beta]}(t) = \begin{cases} \bot & \text{if } G((a, \mathsf{believesToBeFalse}, t)) \in \{\top, \dashv\}, \\ \beta & \text{otherwise.} \end{cases}$$

$$G|_{[a,\vdash,\beta]}(t) = \begin{cases} \vdash & \text{if } G((a, \mathsf{believesToBeUnknown}, t)) \in \{\top, \dashv\}, \\ \beta & \text{otherwise.} \end{cases}$$

$$G|_{[a,\dashv,\beta]}(t) = \begin{cases} \dashv & \text{if } G((a, \mathsf{believesToBeConflicted}, t)) \in \{\top, \dashv\}, \\ \beta & \text{otherwise.} \end{cases}$$

2. $G|_{(E_1 \circ E_2)}(t) = G|_{E_1}(t) \circ G|_{E_2}(t)$.

The reader may wonder why we find believes on statements that are annotated as $\top$ or $\dashv$ instead of only $\top$. The reason is that, regarding the information dimension, a statement annotated with $\dashv$ is equivalent to a statement annotated twice, as $\top$ and $\bot$. Hence, the $\dashv$ annotated statement includes the $\top$ annotated statement.

Example 2. Let G be the $\mathcal{FOUR}$-graph described in Example 1. Then,

1. $G|_{[\mathsf{PopeDI},\top,\vdash]} = \{t_9 \mapsto \top, * \mapsto \vdash\}$. Intuitively, the resulting graph has all the statements that are true according to PopeDI and all unstated statements to be unknown.
2. $G|_{(\mathsf{PopeDI},\circledcirc)} = \{t_9 \mapsto \top, * \mapsto \vdash\}$, $G|_{(\mathsf{Arius},\circledcirc)} = \{t_9 \mapsto \bot, * \mapsto \vdash\}$, and $G|_{(\mathsf{Russell},\circledcirc)} = \{t_1 \mapsto \top, t_{10} \mapsto \bot, * \mapsto \vdash\}$. Intuitively, these are the respective beliefs of PopeDI, Arius, and Russell.

3. $G|_{((\mathsf{PopeDI},\circledcirc)\circledcirc(\mathsf{Arius},\circledcirc))} = \{t_9 \mapsto \dashv, * \mapsto \vdash\}$. Intuitively, PopeDI and Arius are conflicted regarding the nature of Jesus deity.
4. $G|_{((\mathsf{PopeDI},\circledcirc)\circledcirc(\mathsf{Russell},\circledcirc))} = \{t_1 \mapsto \top, t_9 \mapsto \top, t_{10} \mapsto \top, * \mapsto \vdash\}$. Intuitively, PopeDI and Russell are not conflicted regarding the nature of Jesus deity, so the most informative beliefs is assigned to triple t_9.

Definition 5 (Syntax of eSPARQL Filter Formulas). eSPARQL filter formulas *are defined recursively as follows:*

1. *Given two variables* $\texttt{?x}, \texttt{?y} \in \mathbf{V}$ *and two IRIs* $a, b \in \mathbf{I}$*, the expressions* $\texttt{?x} = \texttt{?y}$*,* $\texttt{?x} = a$*,* $a = b$*, and* $\mathrm{bound}(\texttt{?x})$ *are atomic eSPARQL filter formulas.*
2. *Given a* $\mathcal{FOUR}$ *value* α*,* $\mathrm{state}(\alpha)$ *is an atomic eSPARQL filter formula.*
3. *Given two eSPARQL filter formulas* φ *and* ψ*, the expressions* $\neg\varphi$*,* $(\varphi \wedge \psi)$ *and* $(\varphi \vee \psi)$ *are compound eSPARQL filter formulas.*

Definition 6 (Semantics of eSPARQL Filter Formulas). *Given a* $\mathcal{FOUR}$*-relation* R*, a mapping* μ *in the domain of* R*, and a eSPARQL filter formula* φ*, the truth value of* φ *on* μ *is defined recursively as follows:*

1. *If* φ *has the form* $u = v$ *then* $\mu(\varphi) = \mathsf{error}$ *if there is a variable* $\texttt{?x}$ *in* φ *such that* $\texttt{?x} \notin \mathrm{dom}(\mu)$*,* $\mu(\varphi) = \mathsf{true}$ *if replacing every variable* $\texttt{?x}$ *in* φ *with* $\mu(\varphi)$ *leads to a true entity, and* $\mathrm{dom}(\mu)$*, otherwise,* $\mu(\varphi) = \mathsf{false}$*.*
2. *If* φ *has the form* $\mathrm{bound}(\texttt{?x})$ *and* $\texttt{?x} \in \mathrm{dom}(\mu)$*,* $\mu(\varphi) = \mathsf{true}$*; otherwise* $\mu(\varphi) = \mathsf{false}$*.*
3. *If* φ *has the form* $\mathrm{state}(\alpha)$*,* $\mu(\varphi) = \mathsf{true}$ *if* $R(\mu) = \alpha$*; otherwise,* $\mu(\varphi) = \mathsf{false}$*.*
4. *If* φ *is a compound eSPARQL filter formula then* φ *is evaluated with the standard SPARQL three-valued logic semantics of the logical connectives* $\neg$*,* $\wedge$*, and* $\vee$ *for the values* true*,* false*, and* error *(see [16]).*

We define the relation $\mu \models_R \varphi$ *to be true if* $\mu(\varphi) = \mathsf{true}$ *for* R*.*

Intuitively, eSPARQL filter formulas extend SPARQL filter formulas with the ability to check the state of the current mapping.

Definition 7 (Syntax of eSPARQL). *The* syntax of eSPARQL queries *and their* in-scope *variables is defined recursively as follows:*

1. *An RDF-star triple pattern* T *is an eSPARQL query whose in-scope variables are the variables occurring in* T*.*
2. *Given two eSPARQL queries* Q_1 *and* Q_2 *and a* $\mathcal{FOUR}$ *operation* $\circ \in \{\circledcirc, \circledcirc\}$*, the expression* $(Q_1 \circ Q_2)$ *is an eSPARQL query whose in-scope variables are* $\mathrm{inScope}(Q_1) \cup \mathrm{inScope}(Q_2)$*.*
3. *Given two eSPARQL queries* Q_1 *and* Q_2 *with the same set* W *of in-scope variables and a* $\mathcal{FOUR}$ *operation* $\circ \in \{\circledcirc, \circledcirc\}$*, the expression* $(Q_1 \circ Q_2)$ *is an eSPARQL query whose in-scope variables are* W*.*
4. *Given a eSPARQL query* Q*, a* $\mathcal{FOUR}$ *operation* $\circ$*, and a filter formula* φ*, the expression* $(Q\ \mathsf{FILTER}_\circ\ \varphi)$ *is an eSPARQL query with in-scope variables* $\mathrm{inScope}(Q)$*.*

5. *Given an eSPARQL query Q, a $\mathcal{FOUR}$ operator $\circ$, and a finite set of variables $W \subset \text{inScope}(Q)$, the expression* $(\mathsf{SELECT}_\circ\ W\ Q)$ *is an eSPARQL query with in-scope variables W.*
6. *Given an eSPARQL query Q, a filter formula φ, and two $\mathcal{FOUR}$ elements α and β, the expression* $(Q\ \mathsf{MAP}\ (\varphi, \alpha, \beta))$ *is an eSPARQL query with in-scope variables* $\text{inScope}(Q)$.
7. *Given a belief query E, and an eSPARQL query Q such that the in-scope variables of Q do not appear in E, the expression* $(\mathsf{BELIEF}\ E\ Q)$ *is an eSPARQL query with in-scope variables* $\text{var}(E) \cup \text{inScope}(Q)$.

Definition 8 (Semantics of eSPARQL). *The result of evaluating an* eSPARQL query *query Q on a $\mathcal{FOUR}$-graph G is a function $(\!|Q|\!)_G : \Omega_{\text{inScope}(Q)} \to \mathcal{FOUR}$, called $\mathcal{FOUR}$-relation, defined recursively as follows:*

1. *Given an RDF-star triple pattern T and a mapping $\mu \in \Omega_{\text{inScope}(Q)}$, $(\!|T|\!)_G(\mu)$ is $G(t)$, where t is the RDF-star triple pattern resulting from substituting every variable $\texttt{?x} \in \text{inScope}(Q)$ with $\mu(\texttt{?x})$.*
2. $(\!|(Q_1 \mathbin{⍓} Q_2)|\!)_G(\mu) = (\!|Q_1|\!)_G(\mu|_{\text{inScope}(Q_1)}) \mathbin{⍓} (\!|Q_2|\!)_G(\mu|_{\text{inScope}(Q_1)})$.
3. $(\!|(Q_1 \mathbin{⍒} Q_2)|\!)_G(\mu) = (\!|Q_1|\!)_G(\mu|_{\text{inScope}(Q_1)}) \mathbin{⍒} (\!|Q_2|\!)_G(\mu|_{\text{inScope}(Q_1)})$.
4. $(\!|(Q_1 \mathbin{⍗} Q_2)|\!)_G(\mu) = (\!|Q_1|\!)_G(\mu) \mathbin{⍗} (\!|Q_2|\!)_G(\mu)$.
5. $(\!|(Q_1 \mathbin{⍘} Q_2)|\!)_G(\mu) = (\!|Q_1|\!)_G(\mu) \mathbin{⍘} (\!|Q_2|\!)_G(\mu)$.
6. $(\!|(Q_1\ \mathsf{FILTER}_\circ\ \varphi)|\!)_G(\mu) = (\!|Q_1|\!)_G(\mu) \circ \alpha_{\mu \models \varphi}$, *where* $\alpha_{\mu \models \varphi} = Id(\circ)$ *if formula* $\mu \models_R \varphi$ *in the $\mathcal{FOUR}$-relation $R = (\!|Q_1|\!)_G$, and* $\alpha_{\mu \models \varphi} = Abb(\circ)$, *otherwise.*
7. $(\!|(\mathsf{SELECT}_\circ\ W\ Q_1)|\!)_G(\mu) = \circ(\{(\!|Q_1|\!)_G(\mu_1)\ :\ \mu_1|_W = \mu\})$.
8. $(\!|(Q_1\ \mathsf{MAP}\ (\varphi, \alpha, \beta))|\!)_G(\mu) = \gamma$, *where* $\gamma = \alpha$ *if* $\mu \models_{(\!|Q_1|\!)_G} \varphi$, *and* $\gamma = \beta$, *otherwise.*
9. $(\!|(\mathsf{BELIEF}\ E\ Q)|\!)_G(\mu) = (\!|Q|\!)_{G|_{E'}}(\mu|_{\text{inScope}(Q)})$, *where E' is the belief query resulting from replacing every variable $\texttt{?x}$ in E with $\mu(\texttt{?x})$.*

Intuitively, each of the SPARQL operators corresponds to two eSPARQL operators (e.g., AND corresponds to ⍓ and ⍒, and UNION corresponds to ⍗ and ⍘), and each of these operators are evaluated according to the Geerts et al. [6] $\mathcal{K}$-annotated SPARQL algebra by choosing $\mathcal{K}$ to be one of the two $\mathcal{FOUR}$ semirings, namely $\mathcal{FOUR}_{\text{th}}$ and $\mathcal{FOUR}_{\text{in}}$. Except for the operator BELIEF, which changes the context graph where the query is evaluated.

Example 3. Let $G = \{t_1, \ldots, t_8\}$ be the annotated graph mentioned in Table 1. Consider the query $Q = (\mathsf{BELIEF}\ (\texttt{?x}, ⍘)\ (\texttt{?y}, \mathsf{a}, \mathsf{FullDeity}))$. According to Definition 8, the result of query Q is a $\mathcal{FOUR}$-relation with mappings μ whose domain includes the variables ?x and ?y. Given such a mapping μ, $(\!|Q|\!)_G(\mu) = (\!|(\texttt{?y}, \mathsf{a}, \mathsf{FullDeity})|\!)_{G|_{(\mu(\texttt{?x}), ⍘)}}(\mu|_{\{\texttt{?y}\}})$. For example, if $\mu(\texttt{?x}) = \mathsf{PopeDI}$ and $\mu(\texttt{?y}) = \mathsf{Jesus}$, then

$$\begin{aligned}(\!|Q|\!)_G(\mu) &= (\!|(\texttt{?y}, \mathsf{a}, \mathsf{FullDeity})|\!)_{G|_{(\mathsf{PopeDI}, ⍘)}}(\mu|_{\{\texttt{?y}\}}) \\ &= G|_{(\mathsf{PopeDI}, ⍘)}((\mathsf{Jesus}, \mathsf{a}, \mathsf{FullDeity})) \\ &= \top.\end{aligned}$$

It is not difficult to see that if variable ?x had been bound to an individual whose beliefs are not encoded in the graph, then $(\!|Q|\!)_G(\mu) = \bot$ because $G|_{(\mathsf{Jesus},\circledcirc)}$ would have hold no information (i.e., it would have annotated all triples with $\bot$).

6 Use Case Requirements Discussion

In this section, we show a eSPARQL query for each of the use cases U1–U4. Since the notation of eSPARQL queries in an algebraic format is not suitable for end-users, we additionally present how this query can be written as an extension of the user SPARQL syntax.

Use Case U1 Query

$$(\mathsf{BELIEF}\ (\mathsf{PopeDI}, \circledcirc)\ (\texttt{?deity}, \mathsf{a}, \mathsf{FullDeity}))$$

The clause $(\mathsf{BELIEF}\ (\mathsf{PopeDI}, \circledcirc)\ \cdot)$ generates the graph G' consisting of all beliefs of PopeDI. The states of duplicate statements are aggregated with operation $\circledcirc$, and statements that are not mentioned are defined to have state $\vdash$. Then, the nested query $(\texttt{?deity}, \mathsf{a}, \mathsf{FullDeity})$ returns a $\mathcal{FOUR}$-relation R whose domain consists of all mappings $\Omega|_{\texttt{?deity}}$. This domain includes all mappings of the form $\mu_u = \{\texttt{?deity} \mapsto u\}$ where $u \in \mathbf{I}$. For each $u \in \mathbf{I}$, $R(\mu) = G'((u, \mathsf{a}, \mathsf{FullDeity}))$. Since the only belief of PopeDI is indicated in triple t_1, we conclude that $R(\mu)$ is $\top$ if u is Jesus, and $R(\mu)$ is $\vdash$, otherwise (see Example 8).

Observe that the query includes the operator $\circledcirc$. This means that the operations are done over the information lattice. In the user eSPARQL syntax, this is indicated in a simple way by introducing the modifier INFO to the SELECT clause. The next listing shows how the graph G' is specified using the FROM BELIEF clause.

```
SELECT INFO ?deity
FROM BELIEF <PopeDI>
WHERE { ?deity a <FullDeity> }
```

Use Case U2 Query

$$(\mathsf{SELECT}_{\circledcirc}\ \texttt{?deity}\ (((\texttt{?x}, \mathsf{a}, \mathsf{Christian})\ \mathsf{MAP}\ \mathrm{state}(\top) \dashv \vdash) \circledcirc (\mathsf{BELIEF}\ (\texttt{?x}, \circledcirc)\ (\texttt{?deity}, \mathsf{a}, \mathsf{FullDeity}))))$$

The internal clause $Q_2 = (\mathsf{BELIEF}\ (\texttt{?x}, \circledcirc)\ \cdot)$ works similarly to the previous example, but the ground belief queries are constructed over variable ?x. Thus, we need to bind this variable to entities whose belief are encoded in the graph (see Example 8). There are then four possible bindings for variable ?x, namely PopeDI, Arius, Christianity, and Russell. The answers μ_2 of the query Q_2 will be joined with the answers μ_1 of the nested query $Q_1 = ((\texttt{?x}, \mathsf{a}, \mathsf{Christian})\ \mathsf{MAP}\ \mathrm{state}(\top) \dashv \vdash)$. This imposes a further restriction of the

binding for variable ?x (remember that $\otimes$ acts as AND). Only for PopeDI and Arius, the state $(\!|Q_1 \otimes Q_2|\!)_G(\mu_1 \cup \mu_2)$ is not $\bot$. Thus, the ($\mathsf{SELECT}_{\otimes}$?deity $\cdot$) aggregates the beliefs of these two instances of variable ?x with operation $\otimes$. If ?y is bound to Jesus, then for PopeDI the mapping μ_2 is annotated as $\top$, whereas for Arius it is annotated as $\bot$. Hence, the belief of Christians regarding the nature of Jesus is $\dashv$.

In user eSPARQL syntax, this query can be expressed using FROM BELIEF with a variable from within a nested SELECT INFO query.

```
SELECT INFO ?deity
WHERE {
  ?x a <Christian> .
  MAP IF (STATE IS TRUE) TO CONFLICTED ELSE UNKNOWN .
  {
    SELECT INFO ?deity
    FROM BELIEF ?x
    WHERE { ?deity a <FullDeity> }
  }
}
```

Use Case U3 Query

$$((\mathsf{BELIEF}\ ((\mathsf{PopeDI}, \otimes) \otimes (?\mathtt{x}, \otimes))\ (?\mathtt{s}, ?\mathtt{p}, ?\mathtt{o}))\ \mathsf{MAP}\ \mathrm{state}(\dashv)\ \top\ \bot)$$

We first consider the believe query $((\mathsf{PopeDI}, \otimes) \otimes (?\mathtt{x}, \otimes))$, which joins the beliefs of PopeDI with all other ?x using $\otimes$. The $\mathcal{FOUR}$-relation constructed by $(?\mathtt{s}, ?\mathtt{p}, ?\mathtt{o})$ is unconstrained; thus, we obtain the join of the information lattice over all triples. Before projecting to ?x via $\mathsf{SELECT}_{\oplus}$ we use MAP to mark all mappings that include any conflict, which are aggregated by the $\mathsf{SELECT}_{\oplus}$. Finally, we map $\dashv$ to $\top$ and everything else to $\bot$.

In the user eSPARQL syntax, we use SELECT without any modifiers, which defaults to the truth lattice.

```
SELECT ?x
WHERE {
  {
    SELECT INFO ?x
    FROM BELIEF <PopeDI> ?x
    WHERE { ?s ?p ?o }
  }
  MAP IF (STATE IS CONFLICTED) TO TRUE ELSE FALSE
}
```

Use Case U4 Query

$$(\mathsf{SELECT}_{\otimes}\ ?\mathtt{x}\ ((\mathsf{BELIEF}\ (?\mathtt{y}, \otimes)\ (\mathsf{BELIEF}\ (?\mathtt{x}, \otimes)\ (\mathsf{Zeus}, \mathsf{a}, \mathsf{FullDeity})))))$$

The nested query ($\mathsf{BELIEF}\ (?\mathtt{y}, \oplus)\ \cdot$) generates the $\mathcal{FOUR}$-annotated graph of all beliefs by ?y aggregating them on the information lattice. Similarly, the nested query ($\mathsf{BELIEF}\ (?\mathtt{x}, \oplus)\ \cdot$) generates the $\mathcal{FOUR}$-annotated graph of all beliefs by ?x according the beliefs of ?y. Then, the innermost clause $(\mathsf{Zeus}, \mathsf{a}, \mathsf{FullDeity})$

obtains a $\mathcal{FOUR}$-relation R with mappings μ whose in-scope variables are `?y` and `?x`. For example, $R(\{\texttt{?y} \mapsto \mathsf{Russell}, \texttt{?x} \mapsto \mathsf{PopeDI}\}) = \top$ because according to the beliefs of Russell, PopeDI beliefs that (Zeus, a, FullDeity). On the other hand, $R(\{\texttt{?y} \mapsto \mathsf{Arius}, \texttt{?x} \mapsto \mathsf{PopeDI}\}) = \vdash$ because the graph does not contain information about the beliefs of Arius regarding the beliefs of PopeDI. Finally, the clause $(\mathsf{SELECT}_{\Subset}\ \texttt{?x}\ \cdot)$ aggregates the mappings on such a relation R for each instance of the variable `?x`.

In the user eSPARQL syntax, we use a nested SELECT query for each level of nesting in the beliefs.

```
SELECT INFO ?x
WHERE {
  {
    SELECT INFO *
    FROM BELIEF ?y
    WHERE {
      SELECT INFO *
      FROM BELIEF ?x
      WHERE { <Zeus> a <FullDeity> }
    }
  }
}
```

7 Finitely Supported eSPARQL

To implement eSPARQL, the query results must be finitely encoded. As we have already shown, we can encode an infinite function with a finite mapping that has a finite support by encoding the non-zero states only.

Definition 9. *An eSPARQL query Q is said to be* finitely supported *if, for every $\mathcal{FOUR}$-graph with finite support, there exists an element $\alpha \in \mathcal{FOUR}$, called a* zero *for Q and G, such that the set $\{\mu \mid (\!|Q|\!)_G(\mu) \neq \alpha\}$ is finite.*

To know if eSPARQL queries are finitely supported, consider two $\mathcal{FOUR}$-relations R_1 and R_2 with finite support that result of evaluating the respective queries Q_1 and Q_2. That is, there are two $\mathcal{FOUR}$ operations $\circ$ and $\diamond$, such that the sets $\mathrm{supp}_\circ(R_1)$ and $\mathrm{supp}_\diamond(R_2)$ are finite. Given a $\mathcal{FOUR}$ operation $\bullet$, does $R_3 = R_1 \bullet R_2$ have a finite support? To answer this question, lets start figuring what is needed for R_3 to have a finite support for each operator combination of the operators $\circ$, $\diamond$, and $\bullet$.

The first observation is that R_1 and R_2 have infinitely many mappings whose states are respectively $\mathrm{Id}(\circ)$ and $\mathrm{Id}(\diamond)$. If $\bullet$ is a join operator (i.e., $\curlyvee$ or $\Subset$), then we will have infinitely many mappings μ such that $R_3(\mu) = \mathrm{Id}(\circ) \bullet \mathrm{Id}(\diamond)$. Since there is exactly one $\mathcal{FOUR}$-operator, namely $*$ such that $\mathrm{Id}(*) = \mathrm{Id}(\circ) \bullet \mathrm{Id}(\diamond)$, we want to know if $\mathrm{supp}_*(R_3)$ is finite. In this case, the answer is straightforward. If we have a mapping μ' such that $R_3(\mu) \neq \mathrm{Id}(*)$ then it must happen that $R_1(\mu) \neq \mathrm{Id}(\circ)$ or $R_2(\mu) \neq \mathrm{Id}(\diamond)$. Since R_1 and R_2 have finite support, there are finitely many mappings μ satisfying that condition. Thus, $\mathrm{supp}_*(R_3)$ is finite. Hence, for the two operations corresponding to the two lattice join (i.e., the

generalizations of the SPARQL `UNION`), two $\mathcal{FOUR}$-relations with finite support result in a $\mathcal{FOUR}$-relation with finite support.

We next show that if $\bullet$ is a meet operator (i.e., $\oslash$ or $\ominus$), then the resulting $\mathcal{FOUR}$-relation can have no finite support. To this end, consider the query $Q_1 \ominus Q_2$ where Q_1 and Q_2 are two queries with respective in-scope variables `?x` and `?y`. Given a $\mathcal{FOUR}$-annotated graph G, let R_1 be $(\!|Q_1|\!)_G$ and R_2 be $(\!|Q_2|\!)_G$, $\text{supp}_{\oslash}(R_2)$ be finite, $R_1(\{\texttt{?x} \mapsto a\}) = \top$, $R_1(\{\texttt{?x} \mapsto b\}) = \bot$, and assume that there are infinitely many values c such that $R_2(\{\texttt{?y} \mapsto c\}) = \top$. Then,

$$(\!|Q_1 \ominus Q_2|\!)_G(\{\texttt{?x} \mapsto a, \texttt{?y} \mapsto c\}) = \top,$$
$$(\!|Q_1 \ominus Q_2|\!)_G(\{\texttt{?x} \mapsto b, \texttt{?y} \mapsto c\}) = \vdash.$$

Since we can take infinitely many values of c to produce these two different states, we conclude that the answer of query $Q_1 \ominus Q_2$ has no finite support.

Proposition 2. *Every fragment of eSPARQL that includes the operations $\oslash$ or $\ominus$ can include queries that are not finitely supported.*

This negative result is not necessarily an impediment for implementing eSPARQL. In general, since we are interested in statements about the individuals who appear in a knowledge graph, it suffices to consider mappings that range to the active domain of the graph (i.e., individuals that occur in triples in the support of the graph). Since this subset of $\mathbf{I}$ is finite, using the active domain will lead to a query language whose results can be finitely encoded.

8 Related Work

There are many works on four-valued logics [8,15,17], but no one of them considers them for the semantics and query evaluation in SPARQL. Other works annotate SPARQL answers with lattice elements [2,13], but they do not provide a mean to operate with sets of statements encoding beliefs. Arnout et al. [1] consider knowledge graphs with negative facts. However, they do not consider conflicted statements as we did. Works on distributed knowledge contexts are all assuming that knowledge was represented using different ontologies but with the same epistemological status [4,7,9]. Some of them used SPARQL queries as mappings [19], but did not address epistemological status as a key concern. Schenk et al. [18] studies the semantics of trust and caching in the Semantic Web considering the $\mathcal{FOUR}$ structure, however did not consider the problem of querying.

9 Conclusions and Future Work

We presented eSPARQL, a novel approach that allows for the description of epistemic information using RDF-star and formulating epistemic queries using a query language which extends SPARQL-star. This query language is based on

the concrete $\mathcal{FOUR}$ bilattice, but we expect to generalize it to include more general bilattices.

Future work could include the study of different ways to implement eSPARQL. The most direct way is to build on top of a standard SPARQL-star engine. Indeed, the functionality of `FROM BELIEF` clauses to generate a new graph representing people's beliefs can be implemented with SPARQL-star CONSTRUCT queries, which use aggregate operations to compute states of duplicated statements on a set of beliefs. Then, SPARQL-star SELECT queries can be executed on top of the results from these CONSTRUCT queries. Other implementations include: (i) the whole rewriting of a eSPARQL query as a single SPARQL-star SELECT query, and (ii) using specialized indexes and algorithms to implement this query language.

Implementation

We implemented eSPARQL on top of Apache Jena [5], an open source SPARQL-star engine. Our implementation [14] is available under a free software license.

Acknowledgements. This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under the DFG Germany's Excellence Strategy – EXC 2120/1 – 390831618, and the DFG Excellence Strategy – EXC 2075 – 390740016. We acknowledge the support by the Stuttgart Center for Simulation Science (SimTech).

References

1. Arnaout, H., Razniewski, S., Weikum, G., Pan, J.Z.: Wikinegata: a knowledge base with interesting negative statements. Proc. VLDB Endow. **14**(12), 2807–2810 (2021). https://doi.org/10.14778/3476311.3476350
2. Asma, Z., Hernández, D., Galárraga, L., Flouris, G., Fundulaki, I., Hose, K.: NPCS: native provenance computation for SPARQL. In: Proceedings of the ACM on Web Conference 2024. WWW 2024, Singapore, 13–17 May 2024, pp. 2085–2093. ACM (2024). https://doi.org/10.1145/3589334.3645557
3. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: a nucleus for a web of open data. In: Aberer, K., et al. (eds.) ASWC/ISWC -2007. LNCS, vol. 4825, pp. 722–735. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76298-0_52
4. Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., Stuckenschmidt, H.: C-OWL: contextualizing ontologies. In: Fensel, D., Sycara, K., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 164–179. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39718-2_11
5. Foundation, A.S.: Apache jena. https://jena.apache.org/
6. Geerts, F., Unger, T., Karvounarakis, G., Fundulaki, I., Christophides, V.: Algebraic structures for capturing the provenance of SPARQL queries. J. ACM **63**(1), 7:1–7:63 (2016). https://doi.org/10.1145/2810037
7. Ghidini, C., Serafini, L.: Distributed first order logic. Artif. Intell. **253**, 1–39 (2017). https://doi.org/10.1016/j.artint.2017.08.008

8. Grahne, G., Moallemi, A.: A useful four-valued database logic. In: Proceedings of the 22nd International Database Engineering and Applications Symposium. IDEAS 2018, Villa San Giovanni, Italy, 18–20 June 2018, pp. 22–30. ACM (2018). https://doi.org/10.1145/3216122.3216157
9. Grau, B.C., Parsia, B., Sirin, E.: Combining OWL ontologies using epsilon-connections. J. Web Semant. **4**(1), 40–59 (2006). https://doi.org/10.1016/j.websem.2005.09.010
10. Harris, S., Seaborne, A.: SPARQL 1.1 Query language. Techncial report, W3C Recommendation (2013)
11. Hartig, O., Champin, P.A., Kellogg, G., Seaborne, A.: RDF-star and sparQL-star. Technical report, W3C Final Community Group Report (2021)
12. Hayes, P.J., Patel-Schneider, P.F.: RDF 1.1 semantics. Technical report, W3C Recommendation (2014)
13. Hernández, D., Galárraga, L., Hose, K.: Computing how-provenance for SPARQL queries via query rewriting. Proc. VLDB Endow. **14**(13), 3389–3401 (2021). https://doi.org/10.14778/3484224.3484235
14. Pan, X., Hernández, D., Seifer, P., Lämmel, R., Staab, S.: eSPARQL: SPARQL for Epistemic Queries (2024). https://doi.org/10.18419/darus-4344
15. Patel-Schneider, P.F.: A four-valued semantics for terminological logics. Artif. Intell. **38**(3), 319–351 (1989). https://doi.org/10.1016/0004-3702(89)90036-2
16. Pérez, J., Arenas, M., Gutiérrez, C.: Semantics and complexity of SPARQL. ACM Trans. Database Syst. **34**(3), 16:1–16:45 (2009). https://doi.org/10.1145/1567274.1567278
17. Restall, G.: Four-valued semantics for relevant logics (and some of their rivals). J. Philos. Log. **24**(2), 139–160 (1995). https://doi.org/10.1007/BF01048529
18. Schenk, S.: On the semantics of trust and caching in the semantic web. In: Sheth, A., et al. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 533–549. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88564-1_34
19. Schenk, S., Staab, S.: Networked graphs: a declarative mechanism for SPARQL rules, SPARQL views and RDF data integration on the web. In: Proceedings of the 17th International Conference on World Wide Web. WWW 2008, Beijing, China, 21–25 April 2008, pp. 585–594. ACM (2008). https://doi.org/10.1145/1367497.1367577
20. Pellissier Tanon, T., Weikum, G., Suchanek, F.: YAGO 4: a reason-able knowledge base. In: Harth, A., et al. (eds.) ESWC 2020. LNCS, vol. 12123, pp. 583–596. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-49461-2_34
21. Vrandecic, D.: Wikidata: a new platform for collaborative data collection. In: WWW (Companion Volume), pp. 1063–1064. ACM (2012). https://doi.org/10.1145/2187980.2188242