The Fundamentals of Semantic Versioned Querying

Ruben Taelman[1], Hideaki Takeda[2], Miel Vander Sande[1], Ruben Verborgh[1]

[1]IDLab, Department of Electronics and Information Systems, Ghent University – imec
[2]National Institute of Informatics, Sokendai University

Abstract. The domain of RDF versioning concerns itself with the storage of different versions of Linked Datasets. The ability of querying over these versions is an active area of research, and allows for basic insights to be discovered, such as tracking the evolution of certain things in datasets. Querying can however only get you so far. In order to *derive* logical consequences from existing knowledge, we need to be able to *reason* over this data, such as ontology-based inferencing. In order to achieve this, we explore fundamental concepts on *semantic querying* of versioned datasets using ontological knowledge. In this work, we present these concepts as a semantic extension of the existing RDF versioning concepts that focus on syntactical versioning. We remain general and assume that versions do not necessarily follow a purely linear temporal relation. This work lays a foundation for reasoning over RDF versions from a querying perspective, using which RDF versioning storage, query and reasoning systems can be designed.

1. Introduction

RDF versioning has been an active area of research that looks into storage and querying techniques for different versions of Linked Datasets. These versions do not necessarily follow a purely linear temporal relation, as multiple different versions or *branches* of versions could exist at the same time, as opposed to RDF streams [1].

One of the key benefits of RDF is its ability to encode *semantics* into the data through the use of ontologies. This allows *reasoners* to interpret this data, and use this encoded meaning to transform and infer knowledge. Reasoning within static RDF data and temporal RDF streams are already well-established research domains. Reasoning over RDF versions is however still an underexplored domain. Some preliminary work has already been done in the domain of reasoning over RDF versions. In one work for example, the calculation of semantic differences [2] between versions has been investigated. In another work, reasoning with multi-version ontologies [3] was investigated. These works only cover very specific parts of the reasoning demands within RDF versioning.

For instance, given a versioned dataset, it is currently impossible with the existing systems to efficiently find all versions in which a certain fact can be inferred. Furthermore, the domain of combining reasoning and querying —as is done in Ontology-Based Data Access techniques [4, 5]— within the domain of RDF versioning remains unexplored.

In this work, we introduce a general formalization of semantic versioned querying, i.e., reasoning within RDF versioning from the querying perspective, For this, we extend the five foundational versioned query types [6] that were introduced to cover the

retrieval demands within RDF versioning. We formalize concepts such as reasoning within a single version, version differences, and different versions. Furthermore, we present a prototypical implementation of a versioned RDF store that offers basic rule-based reasoning capabilities at query-time. This prototype demonstrates the benefits of semantic versioning, such as finding all versions in which a certain fact can be inferred, and storage space reduction by inferring facts instead of materializing them beforehand.

The aim of this work is to provide a foundation for the future research and development of semantic versioned querying within RDF stores. This will lead to improvements inside domains that require the semantic analysis on Linked Datasets, for example for analyzing concept drift [7] or tracking diseases in biomedical datasets [8] over time.

This article is structured as follows: In the next section, we discuss related work on semantic versioned querying. In Section 3, we discuss the fundamental concepts on RDF versioning. After that, in Section 4, we introduce new foundational semantic versioned query atoms. In Section 5, we present a proof-of-concept implementation of these atoms with a preliminary evalution. Finally, we conclude in Section 6.

2. Related Work

Semantic versioned querying lies somewhere in between the domains of semantic versioning, stream reasoning, and ontology-based data access. In this section, we discuss the related work in these domains.

2.1. Semantic Versioning

In the context of this paper, we consider semantic versioning to be *the logical reasoning over a collection of dataset versions and ontologies*. On the one hand, this concerns the management of multiple versions of datasets [6, 9, 10, 11], and on the other hand, the reasoning over these versions [2, 3].

A lot of research has been done in the area of ontology evolution [12], i.e., the maintenance of ontologies with respect to domain or requirement changes. Some works focus on the *management and maintenance* of multiple ontology versions [13], while other look more into *applying* different ontology versions on datasets [2, 3]. As we focus on *reasoning* with multiple dataset and ontology versions in this work, we discuss the latter.

SemVersion [2] is a system that provides versioning for RDF ontologies. The authors introduce the concept of a *semantic diff* that takes the semantics of an ontology language into account when calculating the difference between two dataset versions. This concept will be explained in more detail in Section 3, after which we generalize it in Section 4 to enable versioning over both the dataset and the ontology.

Huang et al. [3] propose a reasoning framework over a versioned ontology, which is based on a temporal logic approach. They provide a prototypical implementation of their framework as the MORE system. The difference with our approach is that we

assume non-temporal versions, where the relation between versions is not necessarily linear.

2.2. Stream Processing

Within the domain of RDF Stream Processing (RSP) [14], *stream reasoning* is defined as "the logical reasoning in real time on gigantic and inevitably noisy data streams in order to support the decision process of extremely large numbers of concurrent users" [15]. RSP typically uses the concept of *windowing* as a scalability measure to select subsets of data streams to perform reasoning over. This windowing makes RSP similar to RDF versioning, as not only a single dataset has to be taken into account, but multiple different parts or versions of the dataset needs to be processed. Next to this similarity, there are significant differences between the domains of streaming and versioning which elicits a distinction between them. For instance, stream elements are temporally identified and sorted, while versions are not necessarily temporal, such as hash-based identifiers in version control systems (such as Git [16]). Furthermore, streams typically have a high velocity, while versions evolve at a lower rate. For example, DBpedia [17] publishes a new version at a yearly frequency, and the RDF version of npm [18] is being generated every day. Due to these significant differences regarding ordering and velocity, we see the domains of streaming and versioning as distinct, but partially overlapping domains.

2.3. Ontology-Based Data Access

Ontology-Based Data Access (OBDA) [19] is a technique that offers a semantic query interface on top of a non-semantic datasource using semantic mappings that are applied at query-time. Such mappings can for example be defined between RDF and SQL, using OWL ontologies. These datasources are typically incomplete, on top of which semantic mappings can infer additional knowledge through reasoning. In this work, we are mainly concerned with the inference aspect at query-time, which can be referred to as Ontology-Based Query Answering (OBQA) [20]. The SPARQL entailment regimes specification [21] defines several *entailment regimes* that define how such inferences can be achieved at query-time. At the time of writing, no systems exist yet that can offer OBQA on top of *versioned* RDF datasets. Those systems would require new querying capabilities specific to versioning, which will be introduced in the next section.

3. Fundamentals

In this section, we introduce the fundamental concepts on RDF archiving and the semantic diff.

3.1. RDF Archiving

An *RDF archive* [6] has been defined by Fernández et al. as follows:

> An RDF archive graph is a set of version-annotated triples, where a *version-annotated triple (s, p, o):[i]* is an RDF triple (s, p, o) with a label i representing the version in which this triple holds. The set of all RDF triples [22] is defined as $(U \cup B) \times U \times (U \cup B \cup L)$, where U, B, and L, respectively represent the disjoint, infinite sets of URIs, blank nodes, and literals. Finally, an RDF version of an RDF archive A at snapshot i is the RDF graph $A(i) = \{(s, p, o) | (s, p, o):[i] \in A\}$.

For the remainder of this article, we use the shorthand notation A_i to refer to the RDF version $A(i)$.

To cover the retrieval demands in RDF archiving—also known as RDF versioning—, five foundational query types were introduced [6], which are referred to as *query atoms*. These query atoms are based on the RDF data model [22] and SPARQL query language [23]. In these models, a *triple pattern* is defined as $(U \cup V) \times (U \cup V) \times (U \cup L \cup V)$, with V being the infinite set of variables. A set of triple patterns is called a *Basic Graph Pattern*, which forms the basis of a SPARQL query. The evaluation of a SPARQL query Q on an RDF graph G containing RDF triples, produces a bag of solution mappings $[[Q]]_G$.

The five foundational query atoms introduced by Fernández et al. are the following:

1. **Version materialization (VM)** retrieves data using a query Q targeted at a single version A_i.
   Formally: $VM(Q, A_i) = [[Q]]_{A_i}$.
   Example: *Which books were present in the library yesterday?*
2. **Delta materialization (DM)** retrieves query Q's result change sets between two versions A_i and A_j.
   Formally: $DM(Q, A_i, A_j)=(\Omega^+, \Omega^-)$. *With* $\Omega^+ = [[Q]]_{A_i} \setminus [[Q]]_{A_j}$ *and* $\Omega^- = [[Q]]_{A_j} \setminus [[Q]]_{A_i}$.
   Example: *Which books were returned or taken from the library between yesterday and now?*
3. **Version query (VQ)** annotates query Q's results with the versions (of RDF archive A) in which they are valid.
   Formally: $VQ(Q, A) = \{(\Omega, W) \mid W = \{A(i) \mid \Omega=[[Q]]_{A(i)}, i \in N\} \wedge \Omega \neq \emptyset\}$.
   Example: *At what times was book X present in the library?*
4. **Cross-version join (CV)** joins the results of two queries (*Q1* and *Q2*) between versions A_i and A_j.
   Formally: $VM(Q1, A_i) \bowtie VM(Q2, A_j)$.
   Example: *What books were present in the library yesterday and today?*

5. **Change materialization (CM)** returns a list of versions in which a given query Q produces consecutively different results.
Formally: $\{(i, j) \mid i < j, DM(Q, A(i), A(j)) = (\Omega^+, \Omega^-), \Omega^+ \cup \Omega^- \neq \emptyset, \nexists k \in \mathbb{N} : i < k < j\}$.
Example: *At what times was book X returned or taken from the library?*

3.2. Semantic Diff

Völkel et al. introduce the concept of a *semantic diff* [2] that takes the semantics of an ontology language into account when calculating the diff, which is not the case for a regular *structural diff*, which calculates which triples have been added and which ones have been removed. As an example, consider the dataset with two versions from Listing 1. The typical, structural diff just takes the difference between these two versions at triple level, without taking into account the meaning of the triples. The structural diff of this example can be found in Listing 2. A semantic diff on the other hand, takes into account the meaning of the data. For this example, we know that cat is a subclass of animal. Therefore, the removal of Bob being an animal does not actually take place, because it can still be inferred in version 1, as shown in Listing 3.

```
Version 0:
    ex:Bob a ex:Animal.
    ex:Bob foaf:name "Bob".

Version 1:
    ex:Bob a ex:Cat.
    ex:Bob foaf:name "Bob".

Language:
    ex:Cat rdf:subClassOf ex:Animal.
```

Listing 1: A simple example dataset with two versions about Bob the cat, with a separate ontology language.

```
Removed:
    ex:Bob a ex:Animal.

Added:
    ex:Bob a ex:Cat.
```

Listing 2: The structural diff between the two versions in Listing 1.

```
Removed:
Added:
    ex:Bob a ex:Cat.
```

Listing 3: The semantic diff between the two versions in Listing 1.

The *semantic closure* $s_l(A)$ of a set of RDF triples A is the set of all statements that can be concluded from the statements in A under the semantics of the RDF-based ontology language l. A *semantic diff* $d_l(A,B)$ of two sets of RDF triples (A and B) is formally defined by Völkel et al. as $d_l(A,B) = (+_l(A,B), -_l(A,B))$, with $+_l(A, B) = s_l(B) \setminus (s_l(A) \cap s_l(B))$ and $-_l(A, B) = s_l(A) \setminus (s_l(A) \cap s_l(B))$.

4. Semantic Versioned Query Atoms

As discussed in Section 3, there exist five foundational query atoms for querying RDF archives. In this section, we introduce semantic extensions of these query atoms, similar to the structural diff to semantic diff extension introduced by Völkel et al. More concretely, we will extend these five query atoms with parameters for *language versioning*, instead of only *dataset versioning*.

4.1. Versioned Semantic Closure

To remain in line with the definitions on RDF archiving as listed in Section 3, we extend the semantic closure definition by Völkel et al. as follows: *The versioned semantic closure $s(A_i, L_j)$ of a version V_i is the set of all triples that can be inferred from the triples in A_i under the semantics of the RDF-based ontology language L_j.* In this definition, we consider the RDF-based ontology language l to represent an RDF archive as well, for which we use the notation L_j to refer to the RDF version j of the archive L.

4.2. Query Atoms

In this section, we describe the semantic extensions of the five foundational query atoms for querying RDF archives. The atoms that apply to multiple versions (VQ and CM) are subcategorized to handle the different combinations of single and cross-version dataset and ontology versions.

Each extension is described formally, and an example of its usage is given. All examples apply to the use case of a cat shelter that makes use of an evolving ontology of cat species.

The semantic extension of the five versioned query atoms are defined as follows:

1. **Semantic version materialization (S-VM)** retrieves data using a query Q targeted at a single version A_i in a single ontology version L_j.
   Formally: $S\text{-}VM(Q, A_i, L_j) = [[Q]]_{s(A_i, L_j)}$
   Example: *Which African wild cats were present in the shelter yesterday according to last year's classification?*
2. **Semantic delta materialization (S-DM)** retrieves query Q's result change sets between two versions A_i and A_j respectively using ontology version L_k and L_l.
   Formally: $S\text{-}DM(Q, A_i, A_j, L_k, L_l) = (\Omega^+, \Omega^-)$. With $\Omega^+ = [[Q]]_{s(A_i, L_k)} \setminus$

$[[Q]]_{s(A_j, L_l)}$ and $\Omega^- = [[Q]]_{s(A_j, L_{jl})} \setminus [[Q]]_{s(A_i, L_k)}$

Example: *Which cats that were present in the shelter since ten years ago became a different species over the last year?*

3. **Semantic version query (S-VQ)**

    1. **Semantic intermodal and interontological version query (MOS-VQ)** annotates query Q's results with the versions of RDF archive A and ontology L in which they are valid.

    Formally: $S\text{-}VQ(Q, A, L) = \{(\Omega, V) \mid V = \{(A_i, L_j) \mid \Omega=[[Q]]_{s(A_i, L_j)}, i, j \in N\} \wedge \Omega \neq \varnothing\}$

    Example: *At what points in time were there African wild cats in the schelter, and according to the classification of what time?*

    2. **Semantic intermodal version query (MS-VQ)** annotates query Q's results with the versions of RDF archive A in which they are valid according to ontology version L_j.

    Formally: $S\text{-}VQ(Q, A, L_j) = \{(\Omega, V) \mid V = \{A_i \mid \Omega=[[Q]]_{s(A_i, L_j)}, j \in N\} \wedge \Omega \neq \varnothing\}$

    Example: *At what points in time were there African wild cats in the schelter?*

    3. **Semantic interontological version query (OS-VQ)** annotates query Q's results with the versions of ontology L in which they are valid within dataset version A_i.

    Formally: $S\text{-}VQ(Q, A_i, L) = \{(\Omega, V) \mid V = \{L_j \mid \Omega=[[Q]]_{s(A_i, L_j)}, i \in N\} \wedge \Omega \neq \varnothing\}$

    Example: *At what points in time were the cats that are currently in the shelter ever African wild cats?*

4. **Semantic cross-version join (S-CV)** joins the results of two queries (*Q1* and *Q2*) between versions A_i and A_j respectively using ontology version L_k and L_l.

Formally: $S\text{-}CV(Q_1, Q_2, A_i, A_j, L_k, L_l) = S\text{-}VM(Q_1, A_i, L_k) \bowtie S\text{-}VM(Q_2, A_j, L_l)$

Example: *Which African wild cats were in the shelter yesterday (according to last year's classification) and the day before (according to the current classification)?*

5. **Semantic change materialization (S-CM)**

    1. **Semantic intermodal and interontological change materialization (MOS-CM)** returns a list of consecutive archive and ontology versions in which a given query Q produces different results.

    Formally: $S\text{-}CV(Q, A, L) = \{(i, j, k, l) \mid i < j, k < l, S\text{-}DM(Q, A_i, A_j, L_k, L_l) = (\Omega^+, \Omega^-), \Omega^+ \cup \Omega^- \neq \varnothing, \nexists a \in \mathbb{N} : i < a < j, \nexists b \in \mathbb{N} : k < b < l\}$

    Example: *At what times and in which classification did Bob become an African wild cat?*

    2. **Semantic intermodal change materialization (MS-CM)** returns a list of consecutive archive versions in which a given query Q produces different results between two ontology versions.

Formally: $S\text{-}CV(Q, A, L_k, L_l) = \{(i, j) \mid i < j, S\text{-}DM(Q, A_i, A_j, L_k, L_l) = (\Omega^+, \Omega^-), \Omega^+ \cup \Omega^- \neq \emptyset, \nexists\, a \in \mathbb{N} : i < a < j, \nexists\, b \in \mathbb{N} : k < b < l\}$
Example: *At what times did Bob become an African wild cat between last year's and today's classification?*

3. **Semantic interontological change materialization (MS-CM)** returns a list of consecutive language versions in which a given query Q produces different results between two dataset versions.
Formally: $S\text{-}CV(Q, A_i, A_j, L) = \{(k, l) \mid k < l, S\text{-}DM(Q, A_i, A_j, L_k, L_l) = (\Omega^+, \Omega^-), \Omega^+ \cup \Omega^- \neq \emptyset, \nexists\, a \in \mathbb{N} : i < a < j, \nexists\, b \in \mathbb{N} : k < b < l\}$
Example: *In which classification versions did Bob become an African wild cat between yesterday and today?*

4.3. Query Atom Derivations

Based on the semantic query atom extensions that were introduced in last section, we can derive subtypes for the semantic delta materialisation and semantic cross-version join. These subtypes can be used as simplified form of the foundational semantic query atoms.

Semantic delta materialisation The definition of semantic delta materialization (S-DM) is similar to, but more generic than the semantic diff definition by Völkel et al [2]. While the semantic diff only allows versioning on the dataset, our S-DM definition also enables versioning of the ontology. Similarly, Huang et al. [3] introduce a diff that enables versioning of the ontology, but not on the dataset. As such, our semantic delta materialization definition can be seen as a combination of both. Furthermore, we can express these diff methods in terms of S-DM as follows:

- **Intermodal semantic delta materialisation (MS-DM)** is semantic delta materialization of different versions under the same ontology. This corresponds to the diff method of Völkel et al. [2].
  Formally: $MS\text{-}DM(Q, A_i, A_j, L_k) = S\text{-}DM(Q, A_i, A_j, L_k, L_k)$.
- **Interontological semantic delta materialisation (OS-DM)** is semantic delta materialization of the same version under different ontologies. This corresponds to the diff method of Huang et al. [3].
  Formally: $OS\text{-}DM(Q, A_i, L_k, L_l) = S\text{-}DM(Q, A_i, A_i, L_k, L_l)$.

Semantic cross-version join Similarly, we can define the following derivations of the semantic cross-version join:

- **Intermodal semantic cross-version join (MS-CV)** is semantic cross-version join for different versions under the same ontology. Formally, $MS\text{-}CV(Q_1, Q_2, A_i, A_j, L_k) = S\text{-}CV(Q_1, Q_2, A_i, A_j, L_k, L_k)$.
- **Interontological semantic cross-version join (OS-CV)** is semantic cross-version join of the same version under different ontologies. Formally, $OS\text{-}CV(Q_1, Q_2, A_i, L_k, L_l) = S\text{-}CV(Q_1, Q_2, A_i, A_i, L_k, L_l)$.

5. Proof of Concept

In order to provide a baseline of the proposed semantic versioned querying atoms, we provide a prototypical implementation of a subset of the semantic versioned query atoms that were introduced in Section 4. In this section, we describe this system, followed by an evaluation description, and a presentation of the results.

5.1. System

Our prototype is implemented as an additional semantic layer on top of OSTRICH [9], which is a versioned RDF triple store that offers syntactical versioning. As OSTRICH only supports VM, DM and VQ triple pattern queries at the time of writing, we limit our implementation to the semantic extension of these atoms, namely, S-VM, S-DM and S-VQ triple pattern queries.

As can be seen in Fig. 1, our semantic layer internally uses two OSTRICH stores, one for the versioned dataset, and one for the versioned language.

As a backwards rule-based reasoner, this semantic layer infers additional triples for each S-VM, S-DM or S-VQ query. Rules can be provided in the Notation 3 syntax [24] at query time. It does this through the following steps:

1. Select applicable rules for the given triple pattern.
2. Query the dataset for the given triple pattern and version options.
3. Loop until set of inferred triples stops growing.
    1. Determine rules that can infer new triples
    2. Perform backwards reasoning with these rules by querying the dataset and language for the given triple pattern and version options.
    3. Add newly inferred triples to set of triples

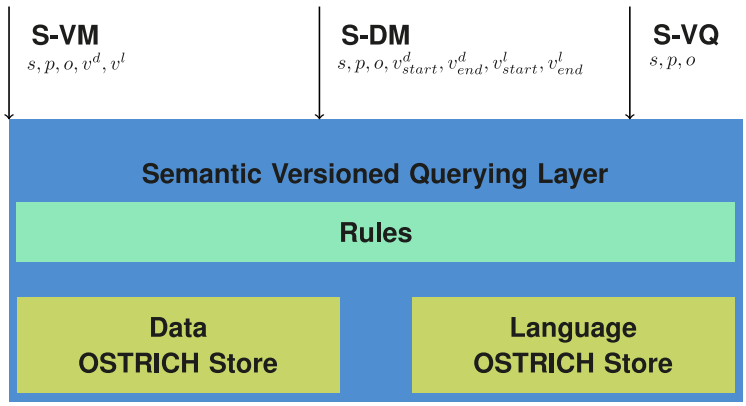

Fig. 1: Architecture overview of our semantic versioned querying layer on top of OSTRICH. The required parameters for the three triple pattern-based queries are indicated, with v^d indicating the version of the dataset, and v^l indicating the version of the language.

The source code of this prototype can be found on GitHub *(https://github.com/rdfostrich/semantic-ostrich)* and is available under the MIT license.

5.2. Evaluation

In order to evaluate the performance of our semantic layer, we executed several queries with inferencing of `rdfs:subClassOf` relationships within the BEAR-B-daily dataset from the BEAR benchmark [6]. The BEAR benchmark evaluates using triple pattern queries, which form the basis of more expressive query evaluation.

To achieve this, we created a derived version of this BEAR-B-daily dataset where we removed all `rdf:type` relationships from instances to classes that can be inferred through `rdfs:subClassOf` relationships for each instance. The (44) triples identifying the subclass relationships were stored in a single version inside the language store. As the BEAR-B-daily dataset does not provide any versioning of the language, our evaluation excludes versioning of the language.

As OSTRICH only supports VM, DM and VQ triple pattern queries, we only evaluate their respective semantic extension, always using the single language version. For S-VM, we query the last version, for S-DM, we query between the first and last version, and for S-VQ, we do an intermodal query using the single language version.

The source code of this evaluation can be found on GitHub *(https://github.com/rdfostrich/semantic-ostrich/blob/master/evaluate.js)*.

5.3. Results

The original BEAR-B-daily dataset contains 48,914 unique triples in 88 dataset versions, while the derived dataset contains 31,761 triples, which is a reduction of 35,07%.

Table 1, Table 2 and Table 3 respectively contain the evaluation results for the S-VM, S-DM and S-VQ queries. The table columns indicate the following:

- *Query*: The subject of the triple pattern that is queried.
- *Original*: Execution time of the query against the original BEAR-B-daily dataset.
- *Reduced*: Execution of the query against the derived BEAR-B-daily dataset, without inference from our semantic layer.
- *Inferred*: Execution of the query against the derived BEAR-B-daily dataset, with inference using our semantic layer.
- *Inference queries*: The number of queries against the OSTRICH store that were performed by the semantic layer.
- *Inferred normalized*: *Inferred* execution time divided by the number of *inference queries*.

The results show that the backwards reasoner within our prototype requires almost eight queries to the OSTRICH stores on average for this dataset. The queries to the OSTRICH stores form the main bottleneck.

Query	Original	Reduced	Inferred	Inference queries	Inferred normalized
dbr:Palazzo_Parisio_(Valletta)	0.56	0.25	2.51	10	0.25
dbr:Singaporean_general_election,_2015	0.47	0.21	2.26	10	0.23
dbr:What_Do_You_Mean?	0.63	0.22	1.76	9	0.20
dbr:Dancing_with_the_Stars_…	0.58	0.23	1.55	6	0.26
dbr:Doctor_Who_(series_9)	0.33	0.15	0.97	6	0.16
dbr:My_Little_Pony…	0.15	0.09	0.94	7	0.13
dbr:2015	0.26	0.12	0.89	6	0.15
Average	*0.42*	*0.18*	*1.55*	*7.71*	*0.20*

Table 1: Execution times in milliseconds for S-VM queries against the last dataset version, using the first language version for an 7 S?? triple patterns.

Query	Original	Reduced	Inferred	Inference queries	Inferred normalized
dbr:Palazzo_Parisio_(Valletta)	0.45	0.21	3.32	10	0.33
dbr:Singaporean_general_election,_2015	0.39	0.18	2.27	10	0.23
dbr:What_Do_You_Mean?	0.34	0.13	2.17	9	0.24
dbr:Dancing_with_the_Stars_…	0.40	0.20	1.06	6	0.18
dbr:Doctor_Who_(series_9)	0.18	0.12	1.14	6	0.19
dbr:My_Little_Pony…	0.12	0.11	2.47	7	0.35
dbr:2015	0.36	0.18	1.84	6	0.31
Average	*0.32*	*0.16*	*2.04*	*7.71*	*0.26*

Table 2: Execution times in milliseconds for S-DM queries between the first and last dataset versions, both using the first language version for 7 S?? triple patterns.

Query	Original	Reduced	Inferred	Inference queries	Inferred normalized
dbr:Palazzo_Parisio_(Valletta)	0.65	0.29	2.48	10	0.25
dbr:Singaporean_general_election,_2015	0.83	0.28	2.16	10	0.22
dbr:What_Do_You_Mean?	0.58	0.17	1.24	9	0.14
dbr:Dancing_with_the_Stars_…	0.43	0.27	0.96	6	0.16
dbr:Doctor_Who_(series_9)	0.26	0.12	0.75	6	0.13
dbr:My_Little_Pony…	0.13	0.09	1.06	7	0.15
dbr:2015	0.24	0.10	1.17	6	0.20
Average	*0.45*	*0.19*	*1.40*	*7.71*	*0.18*

Table 3: Execution times in milliseconds for S-VQ queries for 7 S?? triple patterns.

6. Conclusions

In this work, we introduced fundamental concepts on how to evaluate *semantic* queries over versioned Linked Datasets. For this, we extended existing *structural* versioned query atoms by coupling them with *language versioning* and *reasoning*.

Our wrapper-based prototypical implementation of these semantic extensions shows that semantic querying, i.e., inference at runtime, is able to significantly reduce storage requirements at the cost of an increase in query time. Together with that, it also brings the additional benefit of being able to select the language version(s) for each query, which would otherwise require dataset duplication when no semantic querying layer is present.

As this is merely a wrapper-based prototype, inference is sub-optimal, and a lot of room for improvement exist. In future work, we foresee improvements regarding the runtime inference based on techniques from the world of OBDA. Furthermore, some RDF stream reasoning techniques could potentially be generalized to work for versioned querying. Native implementations of semantic versioning engines could also reduce the overhead of this wrapper-based approach.

The newly introduced foundational semantic versioned query atoms forms a basis for future research and development of semantic versioned querying within RDF stores, and will consequently enable enhanced analysis over multiple Linked Dataset versions.

References

1. Della Valle, E., Ceri, S., Barbieri, D.F., Braga, D., Campi, A.: A first step towards stream reasoning. In: Future Internet Symposium. pp. 72–81. Springer (2008).
2. Völkel, M., Groza, T.: SemVersion: An RDF-based ontology versioning system. In: Proceedings of the IADIS international conference WWW/Internet. p. 44 (2006).

3. Huang, Z., Stuckenschmidt, H.: Reasoning with multi-version ontologies: A temporal logic approach. In: International Semantic Web Conference. pp. 398–412. Springer (2005).
4. Heymans, S., Ma, L., Anicic, D., Ma, Z., Steinmetz, N., Pan, Y., Mei, J., Fokoue, A., Kalyanpur, A., Kershenbaum, A., others: Ontology reasoning with large data repositories. In: Ontology Management. pp. 89–128. Springer (2008).
5. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. In: Journal on data semantics X. pp. 133–173. Springer (2008).
6. Fernández, J.D., Umbrich, J., Polleres, A., Knuth, M.: Evaluating Query and Storage Strategies for RDF Archives. In: Proceedings of the 12th International Conference on Semantic Systems. ACM (2016).
7. Wang, S., Schlobach, S., Klein, M.: Concept drift and how to identify it. Web Semantics: Science, Services and Agents on the World Wide Web. 9, 247–265 (2011).
8. Afgan, E., Baker, D., Van den Beek, M., Blankenberg, D., Bouvier, D., Čech, M., Chilton, J., Clements, D., Coraor, N., Eberhard, C., others: The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update. Nucleic acids research. 44, W3–W10 (2016).
9. Taelman, R., Vander Sande, M., Verborgh, R.: OSTRICH: Versioned Random-Access Triple Store. In: Proceedings of the 27th International Conference Companion on World Wide Web (2018).
10. Meimaris, M., Papastefanatos, G., Viglas, S., Stavrakas, Y., Pateritsas, C., Anagnostopoulos, I.: A Query Language for Multi-version Data Web Archives. Expert Systems. 33, 383–404 (2016).
11. Neumann, T., Weikum, G.: x-RDF-3X: fast querying, high update rates, and consistency for RDF databases. Proceedings of the VLDB Endowment. 3, 256–263 (2010).
12. Zablith, F., Antoniou, G., d'Aquin, M., Flouris, G., Kondylakis, H., Motta, E., Plexousakis, D., Sabou, M.: Ontology evolution: a process-centric survey. The knowledge engineering review. 30, 45–75 (2015).
13. Zekri, A., Brahmia, Z., Grandi, F., Bouaziz, R.: Temporal schema versioning in \tauOWL: a systematic approach for the management of time-varying knowledge. Journal of Decision Systems. 26, 113–137 (2017).
14. Dell'Aglio, D., Della Valle, E., Calbimonte, J.-P., Corcho, O.: RSP-QL semantics: a unifying query model to explain heterogeneity of RDF stream processing systems. International Journal on Semantic Web and Information Systems (IJSWIS). 10, 17–44 (2014).
15. Barbieri, D., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: Stream reasoning: Where we got so far. In: NeFoRS 2010: 4th International Workshop on New Forms of Reasoning for the Semantic Web: Scalable and Dynamic (2010).
16. Torvalds, L., Hamano, J.: Git: Fast version control system. http://git-scm.com (2010).
17. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: Dbpedia: A nucleus for a web of open data. In: The semantic web. pp. 722–735. Springer

(2007).
18. Van Herwegen, J., Taelman, R., Capadisli, S., Verborgh, R.: Describing configurations of software experiments as Linked Data. In: Proceedings of the First Workshop on Enabling Open Semantic Science (SemSci) (2017).
19. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D.F.: The MASTRO system for ontology-based data access. Semantic Web. 2, 43–53 (2011).
20. Ortiz, M.: Ontology Based Query Answering The Story So Far. (2013).
21. Glimm, B., Ogbuji, C.: SPARQL 1.1 Entailment Regimes. W3C, https://www.w3.org/TR/2013/REC-sparql11-entailment-20130321/ (2013).
22. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1: Concepts and Abstract Syntax. W3C, http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/ (2014).
23. Harris, S., Seaborne, A., Prud'hommeaux, E.: SPARQL 1.1 Query Language. W3C, http://www.w3.org/TR/2013/REC-sparql11-query-20130321/ (2013).
24. Berners-Lee, T.: Notation 3, 1998. http://www.w3.org/DesignIssues/Notation3.html (1998).