Time Travel Queries in RDF Archives

Melisachew Wudage Chekol[1], Valeria Fionda[2], and Giuseppe Pirrò[3]

[1] Data and Web Science Group,
University of Mannheim, Germany
mel@informatik.uni-mannheim.de
[2] DeMaCS, University of Calabria, Italy
fionda@mat.unical.it
[3] Institute for High Performance Computing and Networking, ICAR-CNR, Italy
pirro@icar.cnr.it

Abstract. We research the problem of querying RDF archives. In this setting novel data management challenges emerge in terms of support for time-traversing structured queries. We formalize an extension of SPARQL, called SPARQ–LTL, which incorporates primitives *inspired* by linear temporal logic. We give formal semantics for SPARQ–LTL and devise query rewriting strategies from SPARQ–LTL into SPARQL. The usage of SPARQ–LTL allows to gain conciseness and readability when expressing complex temporal queries. We implemented our approach and evaluated query running time and query succinctness.

1 Introduction

Research in the field of archiving policies of Linked Open Data (LOD) opens up new opportunities for the traceability of Semantic Web data over time [13]. Several strands of research (e.g., [11]) have focused on providing primitives to trace whether a dataset or a particular entity has changed, as these functionalities are not natively supported by the SPARQL query language. Dealing with RDF archives poses challenges both in terms of the representation/storage of historical data and the type of query primitives to be provided. Most existing knowledge bases (e.g., DBpedia) allow to query the latest version of data only, although making available historical data in the form of data dumps. In this case, the design of a data model to store data versions as well as infrastructure for query processing is left open. Other efforts (e.g., [8]) have focused on efficient indexing strategies for RDF archives.

Querying historical data is important in many contexts, from analytic tasks where one needs to understand how knowledge has evolved and updated (e.g., pollution levels in a city), to generic exploratory research, where one is interested in posing queries like *"Retrieve the annotations of a gene* ***since*** *the discovery of a particular interaction"* or *"Find players that are* ***now*** *managing some club they* ***played*** *for"*. Typically one can use plain SPARQL to express such queries. However, besides hindering the readability of a query this approach is tedious and error-prone (e.g., one needs to be consistent with variable names). Therefore, a number of proposals came up with extensions of SPARQL both in terms of language primitives and indexing techniques (see Related Work). Despite the plethora of approaches for querying temporal RDF data, we believe that, especially for the case of RDF archives, having a simple approach that neither require to setup complex processing infrastructures nor to learn complex temporal languages can be useful. Therefore, the tenet of this paper is to study *how to facilitate querying historical RDF data on existing SPARQL processors*.

We formalize a powerful extension of SPARQL called SPARQ–LTL, which allows to use a variety of temporal operators inspired by Linear Temporal Logics (LTL) [9] (e.g., `SINCE`, `NEXT`, `PREVIOUS`). As we will show, SPARQ–LTL allows to write concise readable temporal queries in a simple way. In particular, in SPARQ–LTL the relationships between time points are implicit and transparent to the user. To evaluate SPARQ–LTL queries we devise a translation from SPARQ–LTL to SPARQL that allows to readily use our machineries in an elegant and non-intrusive way on existing SPARQL processors.

SPARQ–LTL **by Example.** We now provide some examples of SPARQ–LTL queries.

Example 1. Select footballers who played at least twice for the same club.

```
SELECT ?person WHERE {
  EVENTUALLY{
    ?person :occupation :football_player .
    ?person  :member_of_team ?club .
    NEXT { EVENTUALLY {
      ?person  :member_of_team ?club .
    }}
  } }
```

The SPARQ–LTL query expressing the example makes usage of the `EVENTUALLY` and `NEXT` temporal operators. `EVENTUALLY` checks that eventually (along the whole time line) one has to be a football player, playing for some club (a binding of the variable `?club`). After identifying the timepoint (data version) where the first part of the query has a solution, the usage of `NEXT`, by moving at the next point in the timeline, checks again the same condition. As an example, M. Hughes played for Manchester United both in 1980-86 and 1988-95. Note that the management of timepoints is completely transparent to the user. We now give another example of SPARQ–LTL along with its translation into SPARQL. We assume each historical data version (e.g., DBpedia dump) to be stored in a separate named graph.

Example 2. Find the name of the coach of the Italian national football team after the sacking of Cesare Prandelli.

SPARQ–LTL	**Translation into SPARQL**
	`SELECT ?n WHERE {`
	`{GRAPH <http://data/v5> {`
`SELECT ?n WHERE {`	`dbp:Italy dbpo:coach dbp:Cesare_Prandelli.`
`PAST {`	`GRAPH <http://data/v6> {`
`dbp:Italy dbpo:coach`	`dbp:Italy dbpo:coach ?n.`
`dbp:Cesare_Prandelli.`	`FILTER(?n != dbp:Cesare_Prandelli)}`
`NEXT {`	`} } UNION ...... UNION{`
`dbp:Italy dbpo:coach ?n.`	`GRAPH <http://data//v1> {`
`FILTER`	`dbp:Italy dbpo:coach dbp:Cesare_Prandelli.`
`(?n!=dbp:Cesare_Prandelli)`	`GRAPH <http://data/v2> {`
`} } }`	`dbp:Italy dbpo:coach ?n.`
	`FILTER(?n != dbp:Cesare_Prandelli)`
	`}   } }}`

The previous SPARQ–LTL query makes usage of `PAST` to find when (in which version) C. Prandelli was the coach. Then, via the nested `NEXT` operator executes the innermost part of the query looking for the Italian football team coach in the subsequent version. Assuming to have 6 versions of the data it can be noted that `PAST` makes usage of `UNION` queries over each of the 6 versions v_i, with $i \in \{1, ..., 6\}$, (since the starting version is the current one, i.e., v_6); then, for each v_i, `NEXT` checks in version v_{i+1} (via a `FILTER`) that the coach changed — actually, note that the evaluation of `PAST` can transparently start from the version v_5 since version v_6 does not have a `NEXT` version. The advantage of using SPARQ–LTL can be noted both in terms of succinctness (the SPARQ–LTL query has $\sim$140 characters while the complete translation $\sim$600) and readability. We formalize the translation in Section 3.3.

Contributions and Outline. We make the following main contributions: (i) a formalization of SPARQ–LTL, a temporal extensions of SPARQL that offers a concise and readable syntax for temporal queries; (ii) a formal semantics; (iii) a translation from SPARQ–LTL into SPARQL; (iv) an experimental evaluation along with an analysis of the succinctness of SPARQ–LTL queries.

2 Preliminaries

This section provides some background about the machineries used in this paper.

2.1 RDF and SPARQL

An RDF triple[4] is a tuple of the form $\langle s, p, o\rangle \in \mathbf{I} \times \mathbf{I} \times \mathbf{I} \cup \mathbf{L}$, where $\mathbf{I}$ (IRIs) and $\mathbf{L}$ (literals) are countably infinite sets. An RDF graph G is a set of triples. To query RDF data, a standard query language, called SPARQL, has been defined. The semantics of a SPARQL query is defined in terms of solution mappings. A (solution) mapping μ is a partial function μ: $\mathcal{V} \rightarrow \mathbf{I} \cup \mathbf{L}$. Two mappings, say μ_1 and μ_2, are *compatible* (resp., not compatible), denoted by $\mu_1 \cong \mu_2$ (resp., $\mu_1 \ncong \mu_2$), if $\mu_1(?v) = \mu_2(?v)$ holds (resp., does not hold) for all variables $?v \in \big(\mathrm{dom}(\mu_1) \cap \mathrm{dom}(\mu_2)\big)$. If $\mu_1 \cong \mu_2$ then $\mu_1 \cup \mu_2$ denotes the mapping obtained by extending μ_1 according to μ_2 on all variables in $\mathrm{dom}(\mu_2) \backslash \mathrm{dom}(\mu_1)$. This allows for defining the join, union, difference, and left outer join operations between two sets of mappings M_1, and M_2 as shown below:

$$\begin{aligned}
M_1 \bowtie M_2 &= \{\mu_1 \cup \mu_2 \mid \mu_1 \in M_1, \mu_2 \in M_2 \text{ and } \mu_1 \cong \mu_2\} \\
M_1 \cup M_2 &= \{\mu \mid \mu \in M_1 \text{ or } \mu \in M_2\} \\
M_1 \setminus M_2 &= \{\mu_1 \in M_1 \mid \forall \mu_2 \in M_2, \mu_1 \not\sim \mu_2\} \\
M_1 ⟕ M_2 &= M_1 \bowtie M_2 \cup M_1 \setminus M_2
\end{aligned}$$

The SPARQL semantics uses a function $[\![Q]\!]_G$ that evaluates a query Q on a graph G and gives a multiset (bag) of mappings in the general case.

[4]We do not consider bnodes.

2.2 Linear Temporal Logic

Linear temporal logic (LTL) is an extension of modal logic to formally specify systems and reason over them; here, modalities are temporal operators relating events happening at different time instants over a linearly ordered timeline. Classically, LTL formulas are interpreted over infinite sequences of states. The LTL$_f$ variant [9, 14] considers formulas interpreted over traces of finite length. Given the fact that we *take inspiration* from LTL in the context of dynamic knowledge bases, we will focus on the PLTL$_f$ variant, which is an extension of LTL$_f$ with past modalities [25].

PLTL$_f$ formulas are built over a set V of atomic propositional variables by using the Boolean connectives $\wedge, \vee, \neg$ plus the *future temporal operators* X (`NEXT`), F (`EVENTUALLY`), G (`ALWAYS`), U (`UNTIL`), and the *past temporal operators* Y (`PREVIOUS`), P (`PAST`), H (`ALWAYSPAST`), S (`SINCE`). The meaning of the temporal operators is summarized in Table 1.

PLTL$_f$ Operator	**Meaning**
$\mathsf{X}\, q$	q has to hold in the next state
$\mathsf{F}\, q$	q has to hold in some future state
$\mathsf{G}\, q$	q has to hold in the current and all future states
$q_1 \,\mathsf{U}\, q_2$	q_1 has to hold in all states in the future until there is a state in which q_2 holds
$\mathsf{Y}\, q$	q has to hold in the previous state
$\mathsf{P}\, q$	q has to hold in some past state
$\mathsf{H}\, q$	q has to hold in all past states
$q_1 \,\mathsf{S}\, q_2$	q_1 has to hold starting from the state in the past where q_2 holds

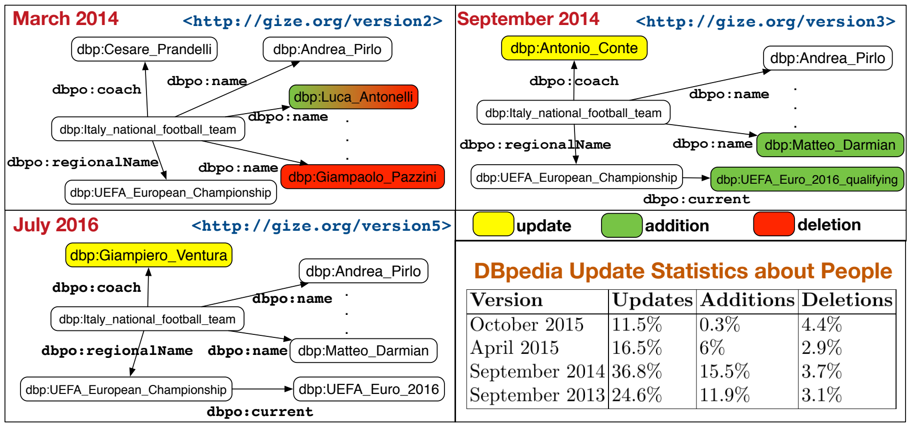
Table 1. Meaning of LTL temporal operators.

2.3 Archiving Policies

Our main goal in this paper is to devise a temporal extension of SPARQL with particular emphasis on an easy (and intuitive) usage of temporal operators and their combinations. Our second goal is to enable querying RDF archives on *existing SPARQL processors*. In what follows we provide an overview about archiving policies for historical RDF data [11].

Independent Versions. The first approach works by keeping *independent versions* of the data. In the case of RDF this could be implemented by assuming that each data version (e.g., DBpedia dumps) is stored in a separate named graph. Consider the data taken from DBpedia reported in Fig. 1. A query ran on March 2014 and asking for the coach of the Italian football team would have returned Cesare Prandelli. The same query ran on September 2014 would have returned Antonio Conte. Query infrastructures allowing to query the latest version of the data, miss the flow of updates. As an example it would *not* be possible to ask queries like *"Find players since C. Prandelli was the coach"*. A simple way to represent historical RDF data using versioning is via RDF quads. An RDF quad (for simplicity, we omit bnodes) is a tuple of the form $\langle s, p, o, c\rangle \in \mathbf{I} \times \mathbf{I} \times (\mathbf{I} \cup \mathbf{L}) \times$

I, where the fourth element of the quad represents the *named graph* to which the triple belongs[5]. Hence, quads that differ for the fourth element only (e.g., white (uncolored) triples in Fig. 1), represent the fact that a triple is present in different versions.



DBpedia Update Statistics about People

Version	Updates	Additions	Deletions
October 2015	11.5%	0.3%	4.4%
April 2015	16.5%	6%	2.9%
September 2014	36.8%	15.5%	3.7%
September 2013	24.6%	11.9%	3.1%

Fig. 1. An excerpt of evolving data from DBpedia.

Tracking the changes. This approach is based on the computation (and storing) of updates or differences (deltas) between versions. It requires additional computational costs for delta propagation which may affect version-focused retrieving operations.

Timestamps. In this case each RDF triple is annotated with its temporal validity. From a practical point of view, there are two main approaches: (i) compression techniques e.g. using selfindexes or delta compression in B+Trees (e.g., [17]); (ii) annotate triples (with time information) only when they are added or deleted.

In this paper we consider the independent versions model to describe our SPARQ–LTL language. Nevertheless, SPARQ–LTL can be also used (with minor changes) with the other archiving models (see Section 6).

3 The SPARQ–LTL Language

We are now ready to introduce SPARQ–LTL, a general language that offers a concise syntax for the writing of temporal queries. We first discuss SPARQ–LTL syntax, then its formal semantics[6] and finally the translation of SPARQ–LTL queries into SPARQL.

3.1 Syntax

Let $\mathcal{V}$ be a set of variables. The syntax of SPARQ–LTL query patterns (QP) is shown in Fig. 2. The language extends SPARQL with constructs *inspired by* the temporal operators of Linear Temporal Logic (LTL) as described in Table 1. In SPARQ–LTL the

[5]For instance, the date in which a version is created

[6]For the sake of readability we focus on set semantics.

complete name of the temporal constructs can be used, e.g. one can write `ALWAYS` instead of G or `ALWAYSPAST` instead of H.

QP ::= **SPARQL operators**
$t = (\mathbf{I} \cup \mathcal{V}) \times (\mathbf{I} \cup \mathcal{V}) \times (\mathbf{I} \cup \mathbf{L} \cup \mathcal{V})$ | QP_1 AND QP_2 | $\{\mathsf{QP}_1\}$ UNION $\{\mathsf{QP}_2\}$ |
$\{\mathsf{QP}_1\}$ MINUS $\{\mathsf{QP}_2\}$ | QP_1 OPT $\{\mathsf{QP}_2\}$ | QP FILTER $\{R\}$ | GRAPH $\mathbf{I} \cup \mathcal{V}$ $\{\mathsf{QP}\}$ |

Temporal operators
X{QP} | F{QP} | G{QP} | $\{\mathsf{QP}_1\}$ U $\{\mathsf{QP}_2\}$ |
Y{QP} | P{QP} | H{QP} | $\{\mathsf{QP}_1\}$ S $\{\mathsf{QP}_2\}$

Fig. 2. Syntax of SPARQ–LTL.

3.2 Semantics

The formal semantics of SPARQ–LTL is shown in Table 2; rules in the top part define the semantics of (standard) SPARQL operators (we focus on set semantics for sake of exposition), while the semantics of the temporal constructs is defined in the bottom part.

$\llbracket t \rrbracket_{G_i}$	$=$	$\{\mu \mid dom(\mu) = var(t)$ and $\mu(t) \in G_i\}$
$\llbracket q_1$ AND $q_2 \rrbracket_{G_i}$	$=$	$\{\mu \mid \mu \in \llbracket q_1 \rrbracket_{G_i} \bowtie \llbracket q_2 \rrbracket_{G_i}\}$
$\llbracket q_1$ UNION $q_2 \rrbracket_{G_i}$	$=$	$\{\mu \mid \mu \in \llbracket q_1 \rrbracket_{G_i} \cup \llbracket q_2 \rrbracket_{G_i}\}$
$\llbracket q_1$ MINUS $q_2 \rrbracket_{G_i}$	$=$	$\{\mu \mid \mu \in \llbracket q_1 \rrbracket_{G_i} \setminus \llbracket q_2 \rrbracket_{G_i}\}$
$\llbracket q_1$ OPT $q_2 \rrbracket_{G_i}$	$=$	$\{\mu \mid \mu \in \llbracket q_1 \rrbracket_{G_i}$ ⟕ $\llbracket q_2 \rrbracket_{G_i}\}$
$\llbracket q$ FILTER $\{R\} \rrbracket_{G_i}$	$=$	$\{\mu \mid \mu \in \llbracket q \rrbracket_{G_i}$ and `eval`(μ, R)`=true`$\}$
$\llbracket$ GRAPH `u` $\{q\} \rrbracket$	$=$	$\{\mu \mid \mu \in \llbracket q \rrbracket_{G_u}\}$
$\llbracket$ GRAPH `?v` $\{q\} \rrbracket$	$=$	$\bigcup_{G_j} \{\mu \mid \mu' \in \llbracket q \rrbracket_{G_j}$ and $\mu = \mu' \cup \{$`?v`$\rightarrow G_j\}\}$
$\llbracket \mathsf{X}\ q \rrbracket_{G_i}$	$=$	$\llbracket q \rrbracket_{G_{i+1}}$
$\llbracket \mathsf{F}\ q \rrbracket_{G_i}$	$=$	$\bigcup_{i \leq j < n} \llbracket q \rrbracket_{G_j}$
$\llbracket \mathsf{G}\ q \rrbracket_{G_i}$	$=$	$\{\mu : \mu \in \llbracket q \rrbracket_{G_i} \bowtie \llbracket q \rrbracket_{G_{i+1}} \bowtie ... \bowtie \llbracket q \rrbracket_{G_n}\}$
$\llbracket\ q_1\ \mathsf{U}\ q_2 \rrbracket_{G_i}$	$=$	$\bigcup_{i \leq j < n} \{\mu_1 \bowtie \mu_2 \mid \mu_2 \in \llbracket q_2 \rrbracket_{G_j}$ and $\mu_1 \in \llbracket q_1 \rrbracket_{G_k}, \forall i \leq k < j\}$
$\llbracket \mathsf{Y}\ q \rrbracket_{G_i}$	$=$	$\llbracket q \rrbracket_{G_{i-1}}$
$\llbracket \mathsf{P}\ q \rrbracket_{G_i}$	$=$	$\bigcup_{0 \leq j \leq i} \llbracket q \rrbracket_{G_j}$
$\llbracket \mathsf{H}\ q \rrbracket_{G_i}$	$=$	$\{\mu : \mu \in \llbracket q \rrbracket_{G_i} \bowtie \llbracket q \rrbracket_{G_{i-1}} \bowtie ... \bowtie \llbracket q \rrbracket_{G_0}\}$
$\llbracket\ q_1\ \mathsf{S}\ q_2 \rrbracket_{G_i}$	$=$	$\bigcup_{0 \leq j \leq i} \{\mu_1 \bowtie \mu_2 \mid \mu_2 \in \llbracket q_2 \rrbracket_{G_j}$ and $\mu_1 \in \llbracket q_1 \rrbracket_{G_k}, \forall j < k \leq i\}$

Table 2. Semantics for SPARQ–LTL

The evaluation function $\llbracket \cdot \rrbracket$ requires a graph G_i (encoding a version of the data) for all rules but those (last two in the top part) that actually allow to evaluate the query on a specific graph. By default the evaluation starts from the current version. The result of a query is a set of mappings that are retrieved by looking into a particular (set of) data version(s) as dictated by the temporal constructs. To define the semantics of temporal operators in SPARQ–LTL we took inspiration from the semantics of LTL [9]. Indeed, the result of a SPARQ–LTL query evaluated in a specific version corresponds to the evaluation of the temporal sub-queries in versions that follow (for *future* operators) or precede (for *past* operators) the given query. For instance, the evaluation of the query pattern EVENTUALLY(F)$\{q\}$ in the version G_i, corresponds to the set of mappings obtained by evaluating q in all versions following G_i. On the other hand, evaluating ALWAYSPAST(H)$\{q\}$ would return the common set of mappings that are given by evaluating q in all the versions preceding G_i.

3.3 Translation into SPARQL

SPARQ–LTL allows to express complex temporal queries in a concise way by incorporating constructs inspired by LTL allowing to capture a variety of temporal aspects both involving the past and the future. The second goal of this paper is to make SPARQ–LTL available into existing SPARQL processors. To fulfill this goal we now discuss a translation procedure from SPARQ–LTL queries into SPARQL queries.

The translation of SPARQ–LTL is outlined in Table 3. The (recursive) translation function $\theta(q, i)$, given a SPARQ–LTL query q and a graph version i produces a SPARQL query by translating the temporal operators via a (set of) pattern(s) evaluated on named graphs (via the SPARQL GRAPH operator) maintaining data versions. The SPARQL query is generated by applying recursively the translation function θ to the sub-queries. We give an additional example by considering data versions (stored in separate named graphs) where dbp:INFT is a shorthand for dbp:Italy_national_football_team.

Example 3. Select players who are playing in the Italian national football team under the current coach and that have played under at least one different coach in the past.

SPARQ–LTL	**Translation into SPARQL**

`SELECT ?p WHERE {`
`  dbp:INFT dbpo:name ?p.`
`  dbp:INFT dbpo:coach ?c1.`
`  PAST{`
`    dbp:INFT dbpo:name ?p.`
`    dbp:INFT dbpo:coach ?c2.`
`    FILTER (?c1!=?c2)`
`  } }`
	`SELECT ?p WHERE {`
	`   dbp:INFT dbpo:name ?p.`
	`       dbp:INFT dbpo:coach ?c1.`
	`   {GRAPH <http://gize.org/v5> { dbp:INFT dbpo:name ?p.`
	`   dbp:INFT dbpo:coach ?c2.`
	`       FILTER (?c1!=?c2) } } UNION`
	`{GRAPH <http://gize.org/v4> { dbp:INFT dbpo:name ?p.`
	`    dbp:INFT dbpo:coach ?c2.`
	`        FILTER (?c1!=?c2) } } UNION`
	`{GRAPH <http://gize.org/v3> { dbp:INFT dbpo:name ?p.`
	`    dbp:INFT dbpo:coach ?c2.`
	`    FILTER (?c1!=?c2)} } UNION`
	`{GRAPH <http://gize.org/v2> { dbp:INFT dbpo:name ?p.`
	`    dbp:INFT dbpo:coach ?c2.`
	`    FILTER (?c1!=?c2) } } UNION`
	`{GRAPH <http://gize.org/v1> { dbp:INFT dbpo:name ?p.`
	`    dbp:INFT dbpo:coach ?c2.`
	`    FILTER (?c1!=?c2)} }  }`

The SPARQL query on the right is automatically generated and can be evaluated on existing processors. Note that the translation requires to look into all previous versions.

SPARQ-LTL	Rewriting
$\theta(t = \langle s, p, o \rangle, i)$	$= t$
$\theta(q_1 \text{ AND } q_2, i)$	$= \theta(q_1, i) \text{ AND } \theta(q_2, i)$
$\theta(q_1 \text{ UNION } q_2, i)$	$= \theta(q_1, i) \text{ UNION } \theta(q_2, i)$
$\theta(q_1 \text{ MINUS } q_2, i)$	$= \theta(q_1, i) \text{ MINUS } \theta(q_2, i)$
$\theta(q_1 \text{ OPT } q_2, i)$	$= \theta(q_1, i) \text{ OPT } \theta(q_2, i)$
$\theta(q \text{ FILTER } \{R\}, i)$	$= \theta(q, i) \text{ FILTER } (R)$
$\theta(\text{ GRAPH } \texttt{v} \{q\}, i)$	$= \{ \text{ GRAPH } \texttt{v} \{\theta(q, v)\}\}$
$\theta(\text{ GRAPH } \texttt{?v}\{q\}, i)$	$= \{ \text{ GRAPH } \langle 0 \rangle \{\theta(q, 0)\} \texttt{ BIND}(0 \texttt{ AS ?v})\} \text{ UNION } \ldots \text{ UNION }$ $\{ \text{ GRAPH } \langle n\text{-}1 \rangle \{\theta(q, n\text{-}1)\} \texttt{ BIND}(n\text{-}1 \texttt{ AS ?v})\}$
$\theta(\mathsf{X}\{q\}, i)$	$= \text{ GRAPH } \langle i\text{+}1 \rangle \{\theta(q, i+1)\}$
$\theta(\mathsf{F}\{q\}, i)$	$= \{ \text{ GRAPH } \langle i \rangle \{\theta(q, i)\}\} \text{ UNION } \{ \text{ GRAPH } \langle i\text{+}1 \rangle \{\theta(q, i\text{+}1)\}\} \text{ UNION} \ldots$ $\text{UNION } \{ \text{ GRAPH } \langle n\text{-}1 \rangle \ \{\theta(q, n\text{-}1)\ \}\}$
$\theta(\mathsf{G}\{q\}, i)$	$= \text{ GRAPH } \langle i \rangle \{\theta(q, i)\} \text{ GRAPH } \langle i\text{+}1 \rangle \{\theta(q, i\text{+}1)\} \ldots \text{ GRAPH } \langle n\text{-}1 \rangle \{\theta(q, n\text{-}1)\}$
$\theta(\{q_1\}\mathsf{U}\{q_2\}, i)$	$= \{ \text{ GRAPH } \langle i \rangle \{\theta(q_2, i)\}\} \text{ UNION } \{ \text{ GRAPH } \langle i\text{+}1 \rangle \{\theta(q_2, i\text{+}1)\} \text{ GRAPH } \langle i \rangle \{\theta(q_1, i)\}\}$ $\text{UNION} \ldots \text{ UNION } \{ \text{ GRAPH } \langle n\text{-}1 \rangle \{\theta(q_2, n\text{-}1)\} \text{ GRAPH } \langle i \rangle \{\theta(q_1, i)\} \ldots$ $\text{GRAPH } \langle n\text{-}2 \rangle \{\theta(q_1, n\text{-}2)\}\}$
$\theta(\mathsf{Y}\{q\}, i)$	$= \text{ GRAPH } \langle i\text{-}1 \rangle \{\theta(q, i\text{-}1)\}$
$\theta(\mathsf{P}\{q\}, i)$	$= \{ \text{ GRAPH } \langle i \rangle \{\theta(q, i)\}\} \text{ UNION } \{ \text{ GRAPH } \langle i\text{-}1 \rangle \{\theta(q, i\text{-}1)\}\} \text{ UNION } \ldots \text{ UNION }$ $\{ \text{ GRAPH } \langle 0 \rangle \{\theta(q, 0)\}\}$
$\theta(\mathsf{H}\{q\}, i)$	$= \text{ GRAPH } \langle i \rangle \{\theta(q, i)\} \text{ GRAPH } \langle i\text{-}1 \rangle \{\theta(q, i\text{-}1)\} \ldots \text{ GRAPH } \langle 0 \rangle \{\theta(q, 0)\}$
$\theta(\{q_1\}\mathsf{S}\{q_2\}, i)$	$= \{ \text{ GRAPH } \langle i \rangle \{\theta(q_2, i)\}\} \text{ UNION } \{ \text{ GRAPH } \langle i\text{-}1 \rangle \{\theta(q_2, i\text{-}1)\} \text{ GRAPH } \langle i \rangle \{\theta(q_1, i)\}\}$ $\text{UNION} \ldots \text{ UNION } \{ \text{ GRAPH } \langle 0 \rangle \{\theta(q_2, 0)\} \text{ GRAPH } \langle 1 \rangle \{\theta(q_1, 1)\} \ldots$ $\text{GRAPH } \langle i \rangle \{\theta(q_1, i)\}\}$

Table 3. Translating SPARQ-LTL into SPARQL. $\theta(q, i)$ is the translation function where q is a query pattern and i is the i-th version. Each version is stored in a separate named graph. $\texttt{GRAPH}\langle 0 \rangle$ denotes the oldest version whereas $\texttt{GRAPH}\langle n - 1 \rangle$ the newest.

4 Related Work

In databases, there exists a vast body of literature on temporal relational databases for developing temporal data models and query languages (cf. the survey [30]). The most prominent query language for temporal databases is TSQL2 [33]. TSQL2 is not a part of the standard SQL, however, the latest standard (SQL:2011) supports the valid (viz. application time) and transaction time models [35].

In the Semantic Web, the introduction of time into RDF has been studied almost one decade ago [22]. Gutierrez et al. [22] studied fundamental problems of temporal RDF graphs such as entailment and outlined a query language allowing to express queries making usage of intervals. Along these lines several other extensions of SPARQL such as τ-SPARQL [34], T-SPARQL [18], tRDF [32] and RDF-TX [17] have been proposed. τ-SPARQL extends SPARQL query patterns with two variables `?s` and `?e` to bind the start time and end time of temporal RDF triples and express temporal queries. The evaluation is done by rewriting τ-SPARQL queries into standard SPARQL queries. T-SPARQL leverages a multi-temporal RDF model where each RDF triple is annotated with a temporal element that represents a set of temporal intervals. T-SPARQL is based on TSQL2 (temporal SQL). The tRDF system builds upon the Gutierrez et al. [22] temporal RDF model. tRDF queries are evaluated using an index (viz. tGrin) based on a strategy that clusters RDF triples using a graphical-temporal distance. RDF-TX [17] offers both a temporal extension of SPARQL and an indexing system based on the compressed multiversion B+ tree approach from relational databases. SPARQL-ST is a query language for spatiotemporal RDF data [31]. It extends SPARQL with spatial and temporal variables. The temporal variables appear in the fourth position of valid time temporal triple patterns (i.e., when temporal triples are represented by quads); and thus, these variables can be mapped into time intervals upon query evaluation. Additionally, SPARQL-ST proposed a new filter operator called TEMPORAL FILTER which supports temporal constraints based on Allen's interval relations [1]. Furthermore, an extension of SPARQL-ST called stSPARQL, using the valid time model, is studied in [5, 24], which uses linear constraints to query valid time spatiotemporal RDF data (stRDF). stSPARQL is implemented and integrated into Strabon[7] that extends SPARQL with a set of temporal functions designed based on Allen's interval algebra. It has also functions for time interval intersection, union, and so on. While stSPARQL is based on Allen's interval algebra, SPARQ-LTL is based on LTL and is able to support stSPARQL's temporal functions via the FILTER operator. A logic-based approach for representing validity time in RDF(S) and OWL 2 is reported in [29]. Based on this approach, the authors extend SPARQL by augmenting basic graph patterns with a number of temporal relations such as *during*, *occurs*, *at*, and so on. No implementation is available for the proposed query language. Our goal is to enable the querying of historical RDF data with two tenets: (i) using *existing SPARQL processors*; (ii) providing a *concise, expressive and readable temporal language*. Surprisingly, despite the plethora of related work, none of the existing approaches can fulfill these desiderata. Approaches like tRDF and RDF-TX have the advantage of using ad-hoc indexing strategies. However, this introduces problems of index construction/maintenance and require non-standard components, thus hindering their applicability on existing SPARQL processors. RDF-TX and tRDF consider a subset of SPARQL (basic graph patterns) and the semantics of the temporal language is only given in terms of SPARQL; we provide semantics for temporal operators. None of these languages (τ-SPARQL, T-SPARQL, tRDF, and RDF-TX) has focused on conciseness and readability of temporal queries, leaving to the user the burden to encode in standard SPARQL the temporal parts. On the contrary, SPARQ–LTL works with an abstract syntax that offers a rich class of (abstract) temporal operators inspired by Linear Temporal Logic [14]. We also want to mention: (i) proposals like tOWL [28] that focus on designing extensions to incorporate time; (ii) temporal conjunctive query answering (e.g., Borgwardt et al.[7]); (iii) translation of SPARQL into LTL (e.g., Mateescu et al.[26, 19], Artale et al. [3, 2]); (iv) approaches like the DBpedia Wayback Machine [10] that allow to retrieve data at a certain timestamp (provided by the user). Our approach differs from (i), (ii), (iii) in the fact that we focus on an extension of SPARQL to write concise and readable temporal queries, and query processing on existing SPARQL processors; as for (iii) we proceed in the opposite direction, expressing LTL operators in SPARQL. Finally (iv), only focus on providing access to an entity at a given time and do not offer any temporal query language. We point other relevant literature in the area [11, 23, 20, 27, 4].

[7]Strabon is a spatiotemporal RDF store `http://www.strabon.di.uoa.gr`

5 Experimental Evaluation

In the experimental evaluation we measured: *(i)* query running time when translating a SPARQ–LTL query; *(ii)* query succinctness.

Datasets. We used both local and remote datasets. We considered data (person data and infoboxes) from the latest 7 versions of DBpedia[8] and loaded each version in a separate named-graph by using Blazegraph[9] as SPARQL processor. All the experiments have been performed on an Intel i5 machine with 8GBs RAM. Results reported are the average of 10 runs. Table 4 reports the size (in millions of triples) of the (local) data loaded in each version along with the percentage of updated (Upds), added (Adds) and removed (Dels) triples between versions.

Version	**Size**	**Upds**	**Adds**	**Dels**
3.6	∼40M	-	-	-
3.7	∼51M	11.2 %	6.3 %	2.1%
3.8	∼59M	16.5%	6%	2.9%
3.9	∼66M	36.8%	15.5%	3.7%
2014	∼78M	23.2%	14.4%	3.9%
2015	∼84M	11.5%	6.3%	7.3%
2016	∼86M	4.4%	3.2%	1.2%

Table 4. Statistics about the versions of DBpedia considered.

Running Time. We used a set of 10 SPARQ–LTL queries including from one to four temporal constructs. In the first experiment, we measured the query time after translating the queries. Running times are reported in Table 5. As it can be observed the running time for all queries is in the order of tens of seconds.

Query	Time (s)	Query	Time (s)
Q1	11	Q6	12
Q2	23	Q7	23
Q3	11	Q8	32
Q4	14	Q9	33
Q5	41	Q10	31

Table 5. Running Time.

We noted that the query time grows almost exponentially with the number of temporal constructs involved. Indeed, each temporal construct requires a certain number of `UNION` queries. The cost in terms of time to load all the version was of about 3h (on a

[8]`http://downloads.dbpedia.org`
[9]`https://www.blazegraph.com`

laptop), with a total storage space for all the versions of about 50 GBs. To give a sense of the amount of redundancy introduced by this approach, we noticed that the $\sim$40% of triples are left untouched along all the versions. Overall, this approach has the advantage of providing an immediate way for querying historical RDF data. The cost that one has to pay consists in loading different data versions plus data redundancy. The level of redundancy that can be afforded depends on the dataset; for datasets of small/medium size like DBpedia this may be bearable.

Query Succinctness. We measured the length, in terms of number of characters, of the SPARQ–LTL queries considered along with their translations. Fig. 3 shows the result of this analysis. It is worth to point out that: *(i)* SPARQ–LTL queries not only are shorter but also more readable; *(ii)* the user deals with a lower number of variables.

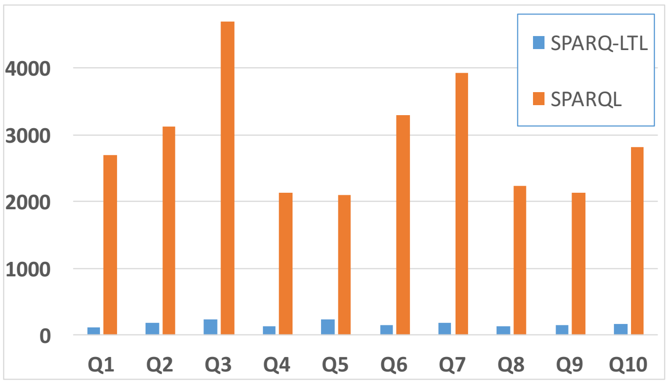


Fig. 3. Query length for SPARQ–LTL and translations.

6 SPARQ–LTL and other Archiving models

We now give a hint of how SPARQ–LTL can be used with other archiving models. In particular we focus on timestamps and leave as a future work a complete treatment of this topic. Timestamps allow to establish the time (or the version) at which a certain event is true. In this setting, a temporal RDF graph is a graph where triples $\langle s, p, o \rangle$ in the graph have a fourth element that represents validity $[v_1, v_2]$, i.e., $(\langle s, p, o \rangle, [v_1, v_2])$. As mentioned in Section 2.3, for triples that do not change at all across versions the validity can be omitted. The syntax of a graph in this setting is usually given by *reifying* temporal facts into non-temporal facts [21]. We represent a temporal triple $(\langle s, p, o \rangle, [v_1, v_2])$ in RDF as shown in Fig. 4.

The advantage of this approach over the independent versions model is that data are stored in one graph, although requiring an effort to generate and keep up to date the validity of triples. In terms of storage space, this approach requires at most (only for triples that change) four triples instead of one.

```
s p _:x.
_:x p o.
_:x :validFrom v1.
_:x :validThrough v2.
```

Fig. 4. Triples with the timestamps model.

The independent versions model can be cast into the timestamp model as follows: (i) start with the oldest data version (v_1) and represent each triple as $\langle s, p, o\rangle$; (ii) for each version v_i, $i > 1$, if $\langle s, p, o\rangle$ (resp., $(\langle s, p, o\rangle, [v_j, \_])$) was in v_{i-1} and is not in v_i add $(\langle s, p, o\rangle, [v_1, v_{i-1}])$ (resp., $(\langle s, p, o\rangle, [v_j, v_{i-1}])$); (iii) if $\langle s, p, o\rangle$ was not in v_{i-1} but is in v_i add $(\langle s, p, o\rangle, [v_i, -])$.

In this setting, the evaluation function used to define the semantics of SPARQ–LTL will not make usage of different (named) graphs (see Table 2) but will make usage of a single graph and leverage the validity information of triples. As an example, the operator EVENTUALLY whose semantics according to the independent versions is

$$[\![\mathsf{F}\ q]\!]_{G_i} = \bigcup_{i \leq j < n} [\![q]\!]_{G_j}$$

with the timestamps is interpreted as:

$$[\![\mathsf{F}\ q]\!]_G^{v_i} = \bigcup_{v_j : v_j \geq v_i} [\![q]\!]_G^{v_j}$$

assuming that timestamps are expressed as integer values and v_i is the version from which $\mathsf{F}\ q$ is evaluated. Similarly, the translation of SPARQ–LTL into SPARQL now takes into account validity information that will be taken into account by using RDF predicates as those in Fig. 4. We give a hint via the following example, which also underlines the conciseness of SPARQ–LTL.

Example 4. Select all footballers that always played with the same football team.

SPARQ–LTL	**Translation into SPARQL**

SELECT ?person WHERE {
EVENTUALLY{
?person :occupation :football_player.
ALWAYS{?person :member_of_team ?club.}
} }
	SELECT ?person WHERE {
	{ ?person :occupation ?x1.
	?x1 :occupation :football_player.
	OPTIONAL{?x1 :validFrom ?t1.}
	OPTIONAL{?x1 :validThrough ?t2.}
	{ ?person :member_of_team ?x2.
	?x2 :member_of_team ?club.
	OPTIONAL{?x2 :validFrom ?t3.}
	OPTIONAL{?x2 :validThrough ?t4.}OPTIONAL
	FILTER(((!BOUND(?t1)&&!BOUND(?t3)) ||
	 (?t3=?t1)) && (!BOUND(?t4)))
	}}}

7 Conclusions and Future Work

We described a novel formal language for querying historical RDF data called SPARQ–LTL. We presented a formal semantics along with translations that makes SPARQ–LTL queries usable in existing SPARQL processors. In this work, we focused on RDF archives with data stored in separate named graphs. As mentioned in Section 6 we are currently working on an extension of the language where archival RDF data are represented via timestamps. As a future work, we will explore the possibility to add other temporal features to the language and perform a more comprehensive experimental evaluation and comparison with related proposals also using benchmarks like BEAR [12]. We will also address temporal coalescing in the valid time model taking into account [6]. The inclusion of preferences [15] and the investigation of temporal navigational languages [16] are also in our research agenda.

References

1. J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
2. A. Artale, R. Kontchakov, A. Kovtunova, V. Ryzhikov, F. Wolter, and M. Zakharyaschev. First-order rewritability of ontology-mediated temporal queries. In *Proc. of the 24th Int. Joint Conf. on Artificial Intelligence (IJCAI'15)*, pages 2706–2712, 2015.
3. A. Artale, R. Kontchakov, V. Ryzhikov, and M. Zakharyaschev. Tractable interval temporal propositional and description logics. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 1417–1423, 2015.
4. S. Auer and H. Herre. A versioning and evolution framework for rdf knowledge bases. In *Perspectives of Systems Informatics*, pages 55–69. Springer, 2006.
5. K. Bereta, P. Smeros, and M. Koubarakis. Representation and Querying of Valid Time of Triples in Linked Geospatial Data. In *The Semantic Web: Semantics and Big Data, 10th International Conference, ESWC 2013, Montpellier, France, May 26-30, 2013. Proceedings*, pages 259–274, 2013.
6. M. H. Böhlen, R. T. Snodgrass, and M. D. Soo. Coalescing in temporal databases. In *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 180–191, 1996.
7. S. Borgwardt, M. Lippmann, and V. Thost. Temporalizing Rewritable Query Languages Over Knowledge Bases. *JWS*, 33:50–70, 2015.
8. A. Cerdeira-Pena, A. Farina, J. D. Fernández, and M. A. Martínez-Prieto. Self-indexing rdf archives. In *Data Compression Conference (DCC), 2016*, pages 526–535. IEEE, 2016.
9. G. D. De Giacomo and M. Y. Vardi. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI*, 2013.
10. J. D. Fernández, P. Schneider, and J. Umbrich. The DBpedia Wayback Machine. In *SEMANTICS*, pages 192–195, 2015.
11. J. D. Fernández, J. Umbrich, A. Polleres, and M. Knuth. Evaluating query and storage strategies for rdf archives. In *Proceedings of the 12th International Conference on Semantic Systems*, pages 41–48. ACM, 2016.
12. J. D. Fernandez Garcia, J. Umbrich, and A. Polleres. Bear: Benchmarking the efficiency of rdf archiving. 2015.
13. V. Fionda and G. Grasso. Linking historical data on the web. In *ISWC Posters & Demonstrations Track*, pages 381–384, 2014.
14. V. Fionda and G. Greco. The Complexity of LTL on Finite Traces: Hard and Easy Fragments. In *AAAI*, 2016.
15. V. Fionda and G. Pirrò. Querying graphs with preferences. In *Proceedings of the 22nd ACM international conference on Information &amp; Knowledge Management*, pages 929–938. ACM, 2013.

16. V. Fionda, G. Pirrò, and M. P. Consens. Extended property paths: writing more sparql queries in a succinct way. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 102–108. AAAI Press, 2015.
17. S. Gao, J. Gu, and C. Zaniolo. RDF-TX: A Fast, User-Friendly System for Querying the History of RDF Knowledge Bases. In *EDBT*, pages 269–280, 2016.
18. F. Grandi. T-SPARQL: A TSQL2-like Temporal Query Language for RDF. In *ADBIS (Local Proceedings)*, pages 21–30. Citeseer, 2010.
19. M. Gueffaz, S. Rampacek, and C. Nicolle. Mapping sparql query to temporal logic query based on nμsmv model checker to query semantic graphs. *International Journal of Digital Information and Wireless Communications (IJDIWC)*, 1(2):366–380, 2012.
20. M. Gueffaz, S. Rampacek, and C. Nicolle. Temporal logic to query semantic graphs using the model checking method. *Journal of Software*, 7(7):1462–1472, 2012.
21. C. Gutierrez, C. Hurtado, and A. Vaisman. Temporal rdf. In *The Semantic Web: Research and Applications*, pages 93–107. Springer, 2005.
22. C. Gutierrez, C. A. Hurtado, and A. Vaisman. Introducing time into rdf. *Knowledge and Data Engineering, IEEE Transactions on*, 19(2):207–218, 2007.
23. U. Khurana and A. Deshpande. Efficient snapshot retrieval over historical graph data. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 997–1008. IEEE, 2013.
24. M. Koubarakis and K. Kyzirakos. Modeling and querying metadata in the semantic sensor web: The model stRDF and the query language stSPARQL. In *The semantic web: research and applications*, pages 425–439. Springer, 2010.
25. F. Laroussinie, N. Markey, and P. Schnoebelen. Temporal logic with forgettable past. In *Logic in Computer Science, 2002. Proceedings. 17th Annual IEEE Symposium on*, pages 383–392. IEEE, 2002.
26. R. Mateescu, S. Meriot, and S. Rampacek. Extending SPARQL with Temporal Logic. Report, 2009.
27. B. McBride and M. Butler. Representing and querying historical information in rdf with application to e-discovery. *HP Laboratories Technical Report, HPL-2009-261*, 2009.
28. V. Milea, F. Frasincar, and U. Kaymak. towl: a temporal web ontology language. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(1):268–281, 2012.
29. B. Motik. Representing and querying validity time in rdf and owl: A logic-based approach. *Web Semantics: Science, Services and Agents on the World Wide Web*, 12:3–21, 2012.
30. G. Özsoyoğlu and R. T. Snodgrass. Temporal and real-time databases: A survey. *Knowledge and Data Engineering, IEEE Transactions on*, 7(4):513–532, 1995.
31. M. Perry, P. Jain, and A. P. Sheth. Sparql-st: Extending sparql to support spatiotemporal queries. In *Geospatial semantics and the semantic web*, pages 61–86. Springer, 2011.
32. A. Pugliese, O. Udrea, and V. Subrahmanian. Scaling RDF with time. In *Proc. of the 17th international conference on World Wide Web*, pages 605–614, 2008.
33. R. T. Snodgrass. *The TSQL2 temporal query language*, volume 330. Springer Science & Business Media, 2012.
34. J. Tappolet and A. Bernstein. Applied temporal RDF: Efficient temporal querying of RDF data with SPARQL. In *European Semantic Web Conference*, pages 308–322. Springer, 2009.
35. F. Zemke. What's new in SQL:2011. *ACM SIGMOD Record*, 41(1):67–73, 2012.