Advanced Temporal Reasoning for Intelligent Traffic Monitoring

Anees ul Mehdi[1,*], Alessandro Oltramari[2]

[1]Corporate Research and Central of Artificial Intelligence, Renningen, Germany
[2]Bosch Center for Artificial Intelligence, Pittsburgh, USA

Abstract

Times series, namely data characterized by temporal information, are a common feature of industrial use cases, especially when considering sensor-based technology. The work presented in this paper focuses on traffic time series collected from stationary cameras, and preprocessed through machine learning algorithms. We introduce the notion of traffic scene, which is central to an ontology of the traffic domain and instrumental to instantiate a knowledge graph for such domain. We then define the formal syntax and semantics of a new language for querying knowledge graphs that is capable of handling temporal information beyond SPARQL expressivity. Finally, we illustrate examples of advanced temporal queries for intelligent traffic monitoring applications.

Keywords

Ontologies, Knowledge Graphs, Temporal Reasoning, Traffic Monitoring

1. Introduction

In this work, we aim to address the limitations of current knowledge graph (KG) approaches in representing and reasoning over dynamic information. In particular, we designed a query language called Temporal Query Language (TQL), which extends SPARQL. The article provides a detailed description of the TQL primitives, clarifying the notion of time assumed in the language, and illustrates an intelligent traffic monitoring application.

2. Problem Description and Setup

Query 1[1] shows a KG excerpt related to traffic objects and their properties. In particular, Line 2 and 3 are asserting that there are two entities of type Car, whereas Line 4 and 5 are describing their speeds (the implicit unit of measure adopted is *miles per hour*).

The instant at which a given information holds can be easily represented in RDF. As an example, we can state that "car 1 has a speed of 22.0 on July 27, 2021 at 14:21". One way of

Ontology Showcase and Demonstrations Track, 9th Joint Ontology Workshops (JOWO 2023), co-located with FOIS 2023, 19-20 July, 2023, Sherbrooke, Québec, Canada.
*Corresponding author.
anees.ulmehdi@de.bosch.com (A. u. Mehdi); alessandro.oltramari@us.bosch.com (A. Oltramari)
https://www.bosch.com/research/about-bosch-research/our-research-experts/alessandro-oltramari/ (A. Oltramari)

[1]tmo is an example prefix.

Query 1: Virtualization query example

```
...
tmo:car_1 rdf:type tmo:Car .
tmo:car_2 rdf:type tmo:Car .
tmo:car_1 tmo:speed 22.0 .
tmo:car_2 tmo:speed 25.0 .
...
```

Query 2: Querying information with time

```
SELECT ?car
WHERE
{
    tmo:car_1  rdf:type tmo:Car .
    tmo:car_1  tmo:associatedWith ?x .
    ?x tmo:hasSpeed ?speed .
    ?x tmo:time ?time .
FILTER(?speed>22.0)
FILTER(?time<"2021-06-27T09:00:00" && ?time>"2021-06-27T21:00:00")
}
```

representing this information in RDF is to use reification[2], whereas tmo:associatedWith is just an auxiliary predicate and _:x is called a blank node. Blank nodes are auxiliary nodes for connecting different entities in a KG.

```
tmo:car_1 tmo:associatedWith _:x .
_:x tmo:hasSpeed 22.0 .
_:x tmo:time "2021-06-27T14:21:00"^^xsd:dateTime
```

Alternatively, we can use RDF-star[3] and represent the same information as following:

```
<<tmo:car_1 tmo:hasSpeed 22.0>>
    tmo:time "2021-06-27T14:21:00"^^xsd:dateTime
```

Regardless of the way we represent temporal information, the question is how to effectively query such information and extract salient dynamic properties from it. When treating time like any other property in RDF, we can use standard query language such as SPARQL. For example, Query 2 asks for all cars with speed greater than 20.0 between 9:00AM to 9:00PM on July 27, 2021. This is a usual interpretation of time. Filters like the one shown in Line 9, allow to query for data that satisfy certain restrictions on the associated time point. In other words, the way we can filter time-associated data is to compare the time points using the usual operators like $, <, =, ! =$ etc. But how to conceptualize and generalize events from time points is left to the human. In case of traffic situations, sometimes we are interested in relating information at different time points and beyond the simple semantics of the aforementioned comparison operators. As an example, suppose we want to check if there is a car idling until a police patrol car arrives at the scene. Such query cannot be formalized in SPARQL, as it would require some sort of a while-loop in order to traverse time-indexed KG triples unless certain conditions are

[2] https://www.w3.org/wiki/RdfReification. Note that we are not using the usual reification approach of associating a triple to a blank node.

[3] https://w3c.github.io/rdf-star/cg-spec/2021-07-01.html

satisfied (appearance of a police vehicle in this example). Query languages like SPARQL lack such a capability. Even though some queries about information over time could be expressed in SPARQL, such queries usually end up to be too complex and prone to error. In this work, our goal is to devise a query language to address these issues. We call the language *temporal query language (TQL)*. TQL basically allows us to query about information while constraining them over time. A detailed description will be provided in upcoming sections. For this language, we take certain assumptions about the notion of time in the target knowledge graph, as described below.

2.1. KG Requirements

The query language we present in this work takes the following assumptions about the target knowledge graph.

Universality of Schema The schema or the ontology part of a given KG graph is assumed to be universal. i.e., we do not allow time to be associated with ontological axioms. In other words, the schema is true at all time points i.e., the axioms or knowledge about a particular domain are not time dependent. Further, we want to avoid computational issues like undecidability of query answering [1].

Temporal Data Not all data need to be temporal. But the parts of the data with temporal information need to be associated with some notion of time. In the previous section, we have seen example how a timestamp can be associated to a triple using the usual datatype xsd:dateTime. Nevertheless, this is not a hard requirement in our case. All we need is some way of ordering information. Further, we need this ordering to be linear i.e., give a time point T in this order, there is at most one successor T' time point of T. A KG $\mathcal{G}$ thus can be perceived as a union of $\mathcal{G}_S, \mathcal{G}_{T_1}, \ldots, \mathcal{G}_{T_n}$ where $\mathcal{G}_S$ represents the schema part of $\mathcal{G}$ along with the triples which are not associated to some time and hence contain no temporal information, whereas G_{T_i} represents the set of triples which are associated with time point T_i for $1 \leq i \leq n$. Without loss of generality, we associate time to a set of triples onward.

3. Use-case

In this work, we apply TQL to improve the analysis of traffic video feeds: the goal is to infer complex behavior from object classes, positions, trajectories recognized by state-of-the-art machine vision systems running on top of stationary cameras. A traffic monitoring ontology has been created, to represent the semantic content or *scene* of each video frame, and to generate a knowledge graph with the annotated data.

3.1. Traffic Monitoring Ontology

Videos collected from different cameras are analyzed frame by frame. For each frame we get information about the objects and their characteristics in it. We describe this information in a KG using different classes and relationships. Note that each frame is recorded at a particular

Query 3: Triples describing an observation about a car in a frame

```
1  tmo:atomicScene1 tmo:composedOf tmo:position1.
2                    tso:TimeStamp "2021-03-03T07:30:44"^^xsd:dateTime;
3  tmo:position1  rdf:type tso:Position;
4                 tmo::hasParticipant tmo:car1;
5                 tmo::hasProperty ?speedProperty1.
6  ?speedProperty1 rdf:type tso:Speed;
7                  tmo:hasValue 20.0.
```

time point. For our use-case, we consider these time points when dealing with dynamic aspect of the domain. The notion of time will be explained in Section 5.

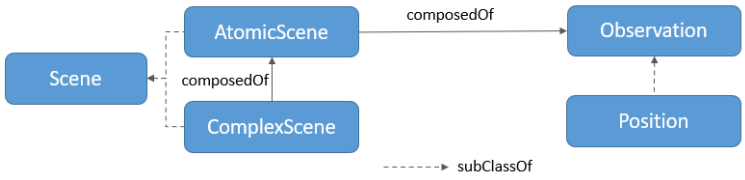

Figure 1: Traffic Monitoring Ontology

Figure 1 shows a snippet of the traffic monotoring ontology (TMO). We next go through the classes and relations in TMO.

AtomicScene and Observation The class AtomicScene is a composition of the class Observation. The elements of Observation represent different types of observations made in a given frame. In our case, the only observations we have are of type Position, which describes a traffic object using some spatial coordinates[4]. However, as an example, if there is a car observed in a frame, triples describing all the relevant information about the car are shown in Query 3. As shown on line 4, Position (Observation) is used to accumulate information about a traffic object (tmo:car1 in this case). What we are describing is that in the given frame we make an observation (of type Position) in which we have a participant (tmo:car1), a property (tmo:Speed) for which we have a value of 20.0. Finally, note that that Line 3 associates a single time point to tmo:atomicScene1. We could actually associate the same time to every triple in Query 3. But since all the triples carry the same time point, we can group them using an instance of AtomicScene.

ComplexScene A complex scene is a composition of atomic scenes. Which atomic scenes constitute a complex scene depends on the pertinent interpretations derived by the use case (see Event) or external constraints.

[4]Including geo-spatial coordinates, bounding boxes and Cartesian coordinates, etc. Note that Figure 1 is only showing a partial view of the ontology, focused on temporal aspects.

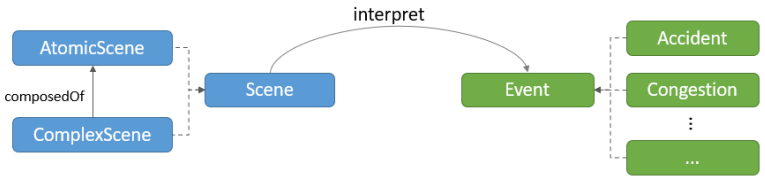

Figure 2: Event as Interpretation

Event Previously we mentioned that complex scenes are composition of atomic scenes. We do not enforce any condition on which atomic scene can be composed into a complex scene. In practice however, we are not interested in random aggregates of atomic scenes. To this end, we use the class Event to represent scenes in a meaningful manner. Events thus can be seen as *semantic tags* we put on scenes (atomic or complex). Figure 3.1 outlines this idea. As an example, a scene can be interpreted as a *congestion* in, say, a village, while it can be interpreted as a *rush hour* in a city. Pictorially, we see in Figure 3.1 the relationship between atomic scenes, complex scenes and events. Different interpretations (events) can be associated to the same complex scene.

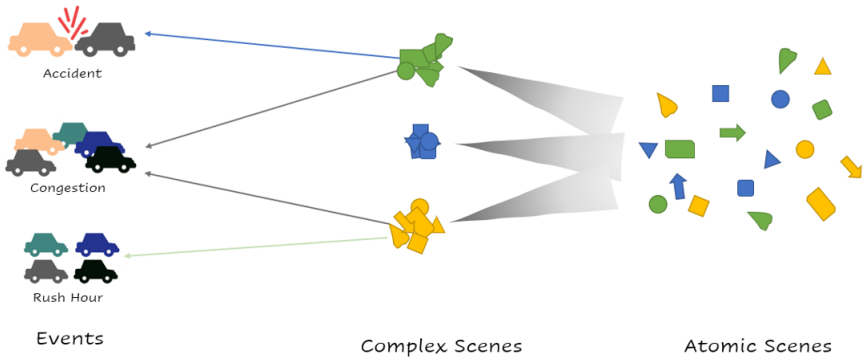

Figure 3: Event Interpretation Examples

After presenting an overview of the relevant part of our KG, we next describe the need for a temporal query language. In the next section, we will see why standard querying is not sufficient in use-cases like traffic monitoring where the data is inherently dynamic and thus involves time as an aspect.

4. Query Languages Limitations

In this section, we describe why standard query languages like SPARQL are not sufficient when dealing with data, such as from traffic monitoring domain, which involves dynamic aspects. This will advocate the idea of devising a more expressive query language which we will introduce in Section 5. To this end, we present the following example scenarios.

Query 4: query checking for accelerating car

```
ASK
WHERE
{
...
tmo:atomicScene_i tmo:composedOf tmo:position_i.
                   tso:TimeStamp ?t_i.
tmo:position_i  rdf:type tso:Position;
                tmo::hasParticipant tmo:car;
                tmo::hasProperty ?speedProperty.
?speedProperty rdf:type tso:Speed;
                 tmo:hasValue ?speed_i.
tmo:atomicScene_j tmo:composedOf tmo:position_j.
                   tso:TimeStamp ?t_j.
tmo:position_j  rdf:type tso:Position;
                tmo::hasParticipant tmo:car;
                tmo::hasProperty ?speedProperty.
?speedProperty rdf:type tso:Speed;
                 tmo:hasValue ?speed_j.
...
FILTER(speed_1<speed_2 && speed_2<speed_3 && ... && speed_19<speed_20)
}
```

Accelerating car: Suppose we are interested in finding out if $Q1$: *a video (from a traffic camera) contains an accelerating car*. For simplicity we imagine that there is only one car (and the same one) in each of the video frames. Suppose there is a video with 20 frames and that these frames are represented as atomic scenes: $A_1, \ldots, A_{20}$ in a KG $\mathcal{G}$. Let $T_1, \ldots, T_{20}$ be the time point associated with $A_1, \ldots, A_{20}$ respectively in the order $T_1 < T_2 < \cdots < T_{19} < T_{20}$. Note that each atomic scene A_i can be represented in $\mathcal{G}$ in the same fashion as in in Query 3. We, thus, are only interested in the speed property of the position for each of the atomic scenes. The given video is a candidate answer to our query if we observe that the speed of the car is increasing as we go from atomic scene A_1 all the way to A_{20}. Retrieving such video from our knoweldge graph can be formalized as a SPARQL query even though it is temporal in nature. All we need to do is to compare the speed corresponding to the position in the atomic scene[5] (frame) at time T_i to speed corresponding to the position in the atomic scene at T_{i+1} where $1 \leq i < 20$. That is, we compare ever consecutive pair of scenes for the speed value. Query 4 shows a snippet of such a query. In general, such queries can be very long and non-trivial. As in our example query, we see that we have to get speed for all the 20 atomic scenes.

Note that not all queries dealing with temporal data can be translated into a SPARQL query (or at least to a single SPARQL query). We call the queries that can be translated into some equivalent SPARQL representation, *simple temporal queries*; we refer to the other queries as *complex temporal queries*. The next example illustrates the latter type.

Congestion ending in an accident: Suppose we are interested in videos where we encounter traffic congestion ending up in an accident. Let's assume by congestion we mean that the cars are not moving at all for at least 10 time points[6]. Further, we identify an accident if two

[5]Frames are represented as instances of the class AtomicScene

[6]We can assume any number of seconds to be one time point.

cars share the same geo-spatial space. In other words, they have overlapping bounding boxes. A video is an answer to this query if (i) there are atomic scenes $S_{T_0}, S_{T_1}, \dots S_{T_n}$ with time points $T_1, T_2, \dots, T_n$ respectively such that all cars have the same position at least in atomic scenes S_{T_i} for $0 \leq i \leq 10$, (ii) no two cars have overlapping position in none of the atomic scenes S_{T_i} for $0 \leq i \leq 10$, and (iii) there is a time point T_k with $10 < k \leq n$ such that there are at least two cars with overlapping positions in the atomic scene S_{T_k}. Further, none of the car changes position nor two car's position overlaps in the scenes S_{T_i} for $10 < i < k$. Note how Condition (i) and (ii) check for a congestion (10 time points) and not happening of an accident. Whereas, Condition (iii) checks for the earliest time point where we encounter an accident. Unlike Condition (i) and (ii), the last condition cannot be translated into a single SPARQL query as we do not know prior how many frames we need to check. This requires some sort of a while-loop which keeps running till the condition is satisfied.

The goal of this work is to answer queries as shown in the above examples. Note that even though the accelerating car example (simple temporal query) can be translated into a SPARQL query, but such queries are large and complex, and thus are prone to error. Next we present a query language which allows expressing simple as well as complex temporal queries.

5. Towards a Formal Query Language

In this section, we define the formal syntax and semantics for the temporal query language we mentioned in the previous section. We first start with some basic notions.

5.1. Preliminaries

All the knowledge graphs we consider here are assumed to be OWL based knowledge graphs[7] which roots in Description Logics (DLs) [3]. We now present some preliminary definitions.

Definition 5.1 (Assertional Triples)**.** A *class assertion triple* is a triple of the following form

[individual] rdf:type *[class].*

A *data property assertion triple* is a triple of the following form

[individual] [data property] [data value].

An *object property assertion triple* is a triple of the following form

[individual] [object property] [individual].

Based on these notions, we next define what we mean by a temporal knowledge graph.

Definition 5.2 (Temporal Knowledge Graph)**.** Let $\mathbb{T}$ be a non-empty set and $<$ be a strict partial order on $\mathbb{T}$. Further, let $\mathcal{T}$ be a set of triples. Then, a *temporal knowledge graph* (or TKG in

[7]OWL is an acronym for Web Ontology Language a W3C standard for describing ontologies [2]. Here we assume basic familarity to RDF, RDFS and OWL.

short) based on $(\mathbb{T}, <)$ and $\mathscr{T}$ is a knowledge graph if there is a set $\mathscr{D}$ of assertional RDF triples with $\mathscr{D} \subseteq \mathscr{T}$ such that

$$\mathscr{D} = \mathscr{D}_{T_1} \cup \mathscr{D}_{T_2} \cup \cdots \cup \mathscr{D}_{T_n}$$

where $T_i \in \mathbb{T}$ for $1 \leq i \leq n$. We represent the knowledge graph by $\mathscr{T}_{(\mathbb{T},<)}$.

The intuition behind this definition is that the set of all assertional triples in a temporal knowledge graph can be partitioned into subsets $\mathscr{D}_{T_i}$ based on some order given by $<$. In practice, we can take $\mathbb{T}$ to be a set of xsd:dateTime literals and $<$ is such that $T_i < T_j$ if T_i is before T_j for two distinct xsd:dateTime times T_i and T_j in $\mathbb{T}$. Note that by definition, we do not require $\mathbb{T}$ to be a set of xsd:dateTime always. All we need is to have a strict partial order $<$ on $\mathbb{T}$. Hence, $\mathbb{T}$ can be set of integers or even strings so far we can establish $<$ amongst its elements[8]. To this end, we define a partial function $\mathsf{time} : \mathscr{T} \mapsto \mathbb{T}$ called *the time association function* such that for any triple $\xi \in \mathscr{D}_{T_i}$ we have $\mathsf{time}(\xi) = T_i$.

Definition 5.2 describes our notion of time. In fact, by time we mean just an order as required by $<$ in the definition of the temporal knowledge graphs. We differ with the usual notion of time (or the natural time) represented by xsd:dateTime. Our definition doesn't restrict us to xsd:dateTime only. In fact, $(\mathbb{T}, <)$ is the only requirement in our definition i.e., all we need is some way of ordering. At first glance, these requirements seem to be a bit confusing for use-cases in practice. However, a knowledge graph where assertional triples can be grouped based on some order is a temporal knowledge graph according to our definition. We will see later why these requirements suffice when we define temporal operators. From now onward, by time we mean an element from some set on which a strict partial order is defined.

Representation of Temporal Knowledge Graph: The representation of time can be a part of the temporal knowledge graph itself. To elaborate this, suppose we have a temporal knowledge graph $\mathscr{T}_{(\mathbb{T},<)}$ and $\mathscr{D}$ is the subset of all assertional triples in $\mathscr{T}$ with

$$D = \bigcup_{i=1}^{n} \mathscr{D}_{T_i}$$

then the two possible representations of time in a temporal knowledge graph are either associating each triple in $\mathscr{D}_{T_i}$ is to T_i using techniques like reification [4] or grouping all the triples in $\mathscr{D}_{T_i}$ in a set and the set is associated to T_i. Without loss of generality we assume the second representation.

5.2. Temporal Query Language (TQL)

As mentioned in the previous section, we restrict the association of time to class assertions, data property assertions and object property assertions only. This restriction allows us to simplify TQL without compromising its expressivity. Further, we avoid issues with computational decidability, which otherwise would be highly probable when dealing with such languages [5].

[8]This ordering can be implicit as well in the sense that we can relate different sets of assertional triples using, say, a property next or successor etc.

Before defining the syntax of TQL, we define some further notions. Here we assume some familiarity with SPARQL query language. We refer the interested reader to [4]. In SPARQL, basic graph patterns (BGPs) are the building blocks of the language. A BGP is an RDF triple where variables are allowed at the position of subject, predicate as well as object. Similar to this, we have the following definitions.

Definition 5.3 (Assertional Basic Graph Pattern)**.** An *assertional basic graph pattern* or (*ABGP* in short) is an assertional RDF triple of one of the following forms:

- *[individual|variable]* rdf:type *[class].*
- *[individual] [data property] [data value|variable].*
- *[individual|variable] [object property] [individual|variable].*

By definition, every assertional triple is an assertional basic graph pattern (without variables) as well as a standard basic graph pattern. ABGPs simply restricts the occurrence of variables at certain positions compared to the standard ones. Hence, each ABGP can be queried against a knowledge using any SPARQL engine.

Definition 5.4 (ABGP Solution)**.** Let γ be an ABGP with $\text{var}(\gamma)$ representing the set of all variables occuring in γ. Then a *solution* for γ in given a knowledge graph $\mathcal{G}$ is a partial function μ from $\text{var}(\gamma)$ to RDF terms in $\mathcal{G}$ such that the assertional triple obtained by simultaneously replacing each variable $x \in \text{var}(\gamma)$ by $\mu(x)$, can be found in $\mathcal{G}$ i.e., $\mu(x) \in \mathcal{G}$.

The notion of solutions can be extended to set of AGBPs in an obvious way: for a set of ABGPs Γ, we say μ is a solution for Γ if and only if μ is a solution for each $\gamma \in \Gamma$.

Note that as shown in 'accelerating car' example in Section 4, a temporal query checks whether a certain relation holds over time. In other words, we want certain facts to hold when traversing from one time point to another. We call such a requirement as a temporal constraint. Hence, a temporal constraint is a condition that need to be satisfied between sets of ABGPs representing factual information where each set corresponds to a time point. In fact, constraints can be applied to the variables occurring in ABGPs due to the fact that variables can represent different values (RDF terms) in a temporal KG at different time points. Hence, a class ABGP can be constrained at the subject position only whereas data and obejct ABGPs can be constrained at both subject as well as object positions. We will see later how the notion of ABGP solutions can be applied to ABGPs with constrained variable.

In practice, we can have different temporal constraints depending on the required conditions.

Variable constrained under constraint $\mathfrak{C}$ is a variable which need to satisfy the conditions enforced by $\mathscr{C}$. $\mathfrak{C}(?x)$ represents a variable $?x$ constrained under $\mathfrak{C}$.

Constrained ABGP is an ABGP such that at least one variable is constrained under exactly one constraint. If the variable is constrained under $\mathfrak{C}$, we say the ABGP is constrained under $\mathfrak{C}$. A constrained ABGP is written as an ABGP except that we write $\mathfrak{C}(x)$ for the constrained variable x. For example, the following is a constrained ABGP where ?*position* is constrained under rigidity (defined below).

$$\text{tmo:car\_1}\ \ \text{tmo:position}\ \ \mathfrak{R}(\text{?position}).$$

We define two temporal constraints of interest that we use in this work. To this end, let $\mathscr{T}_{(\mathbb{T},<)}$ be a temporal KG[9] with $\mathscr{T} = \mathscr{G} \cup \mathscr{D}$ where

$$\mathscr{D} = \mathscr{D}_{T_1} \cup \mathscr{D}_{T_2} \cup \cdots \cup \mathscr{D}_{T_n}$$

and $T_i \in \mathbb{T}$ for $1 \leq i \leq n$. Then,

Definition 5.5. (Rigidity) The Rigidity constraint is represented by $\mathfrak{R}$ and requires the values of the variables not to change when moving from one time point to another. Formally, we say *rigidity is existentially satisfied for a constrained ABGP* χ in the temporal knowledge graph $\mathscr{T}_{(\mathbb{T},<)}$ for time T to T' if and only if there is a solution μ for χ in $\mathscr{G} \cup \mathscr{D}_T$ and a solution μ' for χ in $\mathscr{G} \cup \mathscr{D}_{T'}$ such that for each $x \in \mathsf{var}(\chi)$ constrained under rigidity we have that $\mu(x) = \mu'(x)$. We represent existential rigidity by $\mathfrak{R}_E$.

We say *rigidity is universally satisfied for* χ in the temporal knowledge graph $\mathscr{T}_{(\mathbb{T},<)}$ for time T to T' if and only if for every solution μ for χ in $\mathscr{G} \cup \mathscr{D}_T$, there is a solution μ' for χ in $\mathscr{G} \cup \mathscr{D}_{T'}$ such that for each $x \in \mathsf{var}(\chi)$ constrained under rigidity we have that $\mu(x) = \mu'(x)$. We represent universal rigidity by $\mathfrak{R}_U$.

Note that rigidity requires both mappings μ and μ' to be solutions for χ with the additional requirement that the constrained variable (under rigidity) in χ is mapped to the same value by both mappings. The intuition behind rigidity constraint is that when moving from time T to T', the value for the constrained variable does not change and interprets the triple *rigidly* for the variables constrained under rigidity. Sometimes we intend to allow for minor changes in the values of variables. We define the so-called likelihood constraint. Note that such a constraint makes sense in case of data property assertional triples as we can enforce likelihood on the values of the data property only. The amount of change for data values can be measured using some threshold function i.e., the satisfaction of likelihood constraint relies on certain threshold. We would like to mention here that the definition of threshold functions as well as threshold values can be defined as per use-case. This adds a great flexibility in controlling change over time.

Definition 5.6 (Likelihood)**.** Let ξ be a data property ABGPs and let D be the range of the data property in ξ. In other words, the object position in ξ is either a variable or a value from D. Further let $x \in \mathsf{var}(\xi)$ be the variable at object position. Let τ be a threshold function on D i.e., $\tau : \mathsf{D} \times \mathsf{D} \mapsto \{\mathsf{true}, \mathsf{false}\}$. We say *likelihood is satisfied for* ξ in $\mathscr{T}_{(\mathbb{T},<)}$ given τ for time T to T' if and only if there is a solution μ for ξ in $\mathscr{G} \cup \mathscr{D}_T$ and a solution μ' for ξ in $\mathscr{G} \cup \mathscr{D}_{T'}$ such that $\tau\big(\mu(x), \mu'(x)\big) = \mathsf{true}$. In other words, x is mapped to two values, say, v_1 and v_2 by μ and μ' respectively such that $\tau(v_1, v_2)$ yields true.

[9]Here $\mathscr{G}$ represents the non-temporal part of the knowledge graph. This will be our assumption onward.

From now onward we assume there is always a given threshold function when we consider constrained ABGP where the variable is constrained under likelihood, unless stated otherwise. Note that the notion of universality and existentiality does not make sense in case of likelihood constrained.

Example of Constrained ABGPs: Consider the following ABGP:

$$\chi : \quad \text{tmo:car\_1 tmo:Speed } \mathfrak{L}(?\text{speed}).$$

Here we suppose that the following threshold function is given for $\mathfrak{L}$(?speed)

$$\tau(n, n') = \begin{cases} \text{true} & \text{if } n' - n \leq 1.5 \\ \text{false} & \text{otherwise} \end{cases}$$

Let μ and μ' be solutions for χ in $\mathscr{G} \cup \mathscr{D}_T$ and $\mathscr{G} \cup \mathscr{D}_{T'}$ respectively such that $\mu(?\text{speed}) = 15$ and $\mu'(?\text{speed}) = 18$. We thus see that likelihood (with the given threshold function) is satisfied for χ in $\mathscr{T}_{(\mathbb{T},<)}$ for T to T' as both μ and μ' are solutions as well as $\tau(18, 17) = \text{true}$ since $18 - 15 = 3 \leq 1.5$.

The notion of temporal constraint allows us to enforce certain conditions to be met by solutions when moving over time in a knowledge graph. In other words, temporal constraints are ways to monitor how information in knowledge alters over time. However, *temporal operators* allow us to traverse over time points. There can be different temporal operators. While temporal constraints enforces conditions that need to be satisfied over time, the temporal operators govern what time points need to be considered when applying these constraints.

5.2.1. Syntax

Standard temporal logic allows for different temporal operators. We, however, we define two temporal operators namely, $\mathbf{N}_n$ and $\mathbf{U}$ for defining TQL.

Definition 5.7 (Syntax of TQL)**.** Let $\mathscr{T}_{(\mathbb{T},<)}$ be a temporal knowledge graph with $\mathscr{T} = \mathscr{G} \cup \mathscr{D}$ where $\mathscr{D} = \mathscr{D}_{T_1} \cup \mathscr{D}_{T_2} \cup \cdots \cup \mathscr{D}_{T_n}$. Let $\mathscr{X}$ be a set of ABGPs and $\mathscr{X}_{\mathfrak{C}}$ be a set of constrained ABGPs. Then,

I. any subset of $\mathscr{X}$ is a temporal query.

II. if $\Lambda \subseteq \mathscr{X}$, $\Gamma \subseteq \mathscr{X} \cup \mathscr{X}_{\mathfrak{C}}$ such that no variable in Γ is constrained under more than one constraint and Λ' is non-empty set with $\Lambda' \subseteq \mathscr{X}$, then
   - $\Lambda\ \mathbf{N}_n(\Gamma)$ for $n \geq 1$ is a temporal query called $\mathbf{N}_n$ temporal query.
   - $\Lambda.\Gamma\mathbf{U}\Lambda'$ is a temporal query called $\mathbf{U}$ query.
   - $\Lambda\mathbf{N}_n(\Gamma)\mathbf{U}\Lambda'$ is a temporal query where $n \geq 1$. We call such query as $\mathbf{N}_n\mathbf{U}$ query.

Note how constrained ABGPs are used along with the temporal operators. For simplicity, we restrict to non-recursive occurrences of the temporal operators in temporal queries. Nevertheless, the language can easily be extended to cover more complex queries.

5.2.2. Semantics

In Definition 5.3, we have mentioned that assertional basic graph patterns (ABGPs) are by very definition basic graph patterns as in SPARQL. Also, in Definition 5.4, we have seen the notion of ABGP solution which again is what we have in standard SPARQL. Note that, similar to SPARQL, a set of ABGPs can be seen as a query. Thus, for a given set of ABGPs Φ and a knowledge graph $\mathscr{T}$, a solution for Φ in $\mathscr{T}$ is simply answer to Φ. Based on this, we next define the notion of TQL query answers.

Definition 5.8. Suppose $\mathscr{T}_{(\mathbb{T},<)}$ is a temporal KG with $\mathscr{T} = \mathscr{G} \cup \mathscr{D}$ where $\mathscr{D} = \mathscr{D}_{T_1} \cup \mathscr{D}_{T_2} \cup \cdots \cup \mathscr{D}_{T_n}$ and $T_i \in \mathbb{T}$ for $1 \leq i \leq n$. Further, Let $\mathscr{X}$ be a set of ABGPs and $\mathscr{X}_{\mathfrak{C}}$ be a set of constrained ABGPs. An answer/solution to q in $\mathscr{T}_{(\mathbb{T},<)}$ is defined as following:

- $\underline{q = \{\chi_1, \dots, \chi_k\} \subseteq \mathscr{X}}$
  a mapping μ from $\mathrm{var}(\chi)$ to RDF terms is a solution or an answer to q in $\mathscr{T}_{(\mathbb{T},<)}$ if and only if $\mu(\chi_i)$ is a match in $\mathscr{T}_{(\mathbb{T},<)}$ for $1 \leq i \leq k$. We say μ is a solution or an answer for q in $\mathscr{T}_{(\mathbb{T},<)}$. Note that in this case, we have no constraint nor time to consider as the query is just a set of ABGPs. It, thus can be seen as a SPARQL query.

- $\underline{q = \Lambda\, \mathbf{N}_n(\Gamma) \text{ for } \Lambda \subseteq \mathscr{X}, \Gamma \subseteq \mathscr{X} \cup \mathscr{X}_{\mathfrak{C}} \text{ for } n \geq 1}$
  there are time points $T_0, \dots T_n$ and mappings $\mu_0, \dots, \mu_n$ such that: μ_0 is a solution for Λ in $\mathscr{G} \cup \mathscr{D}_{T_0}$, μ_i is a solution for Γ in $\mathscr{G} \cup \mathscr{D}_{T_i}$ for $0 \leq i \leq n$, and the constraints in $\xi \in \Gamma$ are satisfied for ξ in $\mathscr{T}_{(\mathbb{T},<)}$ for T_i to T_{i+1} where $0 \leq i < n$.

  The *solution to μ for q* is obtained by combining all above μ_i while replacing in constrained variable x by $x^{(i)}$ i.e., if x is a non-constrained variable in q and $x \mapsto \mu_i(x)$ is in μ_i then it is replaced by $x^{(i)} \mapsto \mu_i(x)$ in μ. In simple words, we create copies of the non-constrained variables for each time point.

- $\underline{\Lambda . \Gamma \mathbf{U} \Lambda' \text{ for } \Lambda \subseteq \mathscr{X}, \Gamma \subseteq \mathscr{X} \cup \mathscr{X}_{\mathfrak{C}} \text{ and a non-emtpy set } \Lambda' \in \mathscr{X}}$
  there are time points $T_0, T_1, \dots$ and mappings $\mu_0, \mu_1, dots$, and there exists $l \geq 0$ such that: μ_0 is a solution for Λ in $\mathscr{G} \cup \mathscr{D}_{T_0}$, μ_i is a solution for Γ in $\mathscr{G} \cup \mathscr{D}_{T_i}$ for $0 \leq i < l$, each constraint in constrained ABGP $\xi \in \Gamma$ is satisfied for ξ in $\mathscr{T}_{(\mathbb{T},<)}$ for T_i to T_{i+1} for $0 \leq i < l-1$, there is no solution for Λ' in $\mathscr{G} \cup \mathscr{D}_{T_i}$ for $0 \leq i < l$, and μ_l is a solution for Λ' in $\mathscr{G} \cup \mathscr{D}_{T_l}$.

- $\underline{\Lambda \mathbf{N}_n(\Gamma) \mathbf{U} \Lambda' \text{ for } \Lambda \subseteq \mathscr{X}, \Gamma \subseteq \mathscr{X} \cup \mathscr{X}_{\mathfrak{C}} \text{ and a non-emtpy set } \Lambda' \in \mathscr{X}}$
  there are time points $T_0, \dots, T_n, T_{n+1}, \dots$ and mappings $\mu_0, \dots, \mu_n, \mu_{n+1}, \dots$ such that: μ_0 is a solution for Λ in $\mathscr{G} \cup \mathscr{D}_{T_0}$, μ_i is a solution for Γ in $\mathscr{G} \cup \mathscr{D}_{T_i}$ for $0 \leq i \leq n$, the constraints in $\xi \in \Gamma$ are satisfied for ξ in $\mathscr{T}_{(\mathbb{T},<)}$ for T_i to T_{i+1} where $0 \leq i < n$, there is a time point $l \geq n$ such that μ_i is a solution for Γ in $\mathscr{G} \cup \mathscr{D}_{T_i}$ for $n \leq i < l$, the constraints in $\xi \in \Gamma$ are satisfied for ξ in $\mathscr{T}_{(\mathbb{T},<)}$ for T_i to T_{i+1} where $n \leq i < l-1$, there is no solution μ' for Λ' in $\mathscr{G} \cup \mathscr{D}_{T_i}$ for $n \leq i \leq l-1$, and μ_l is a solution for Λ' in $\mathscr{G} \cup \mathscr{D}_{T_l}$.

Note that the temporal operator $\mathbf{N}_n$ allows us to iterate over a fix number of time points n and check whether a temporal constraint is satisfied or not. Meanwhile, the temporal operator $\mathbf{U}$ checks the validity of a temporal constraint for as long as certain condition is satisfied. In our case, this condition corresponds to the origination of some new facts represented by the set Λ'. Let's consider an example of a temporal query from traffic monitoring domain.

An Accelerating Car Ending in an Accident Suppose a car encounters an accident if it's bounding box overlaps with that of another car. Then, the query can be formalized as

$$\{\text{?scene rdf:type tmo:Scene.} \quad \mathfrak{R}_E(\text{?car}) \text{ rdf:type tmo:Car.} \quad \text{?car tmo:hasSpeed } \mathfrak{L}(\text{?speed}).\}$$
$$\mathbf{U}$$
$$\{\text{?car tmo:hasPosition ?position.} \quad \text{car2 rdf:type tmo:car.} \quad \text{?car2 tmo:hasPosition ?position2.} \quad \text{?position tmo:overlaps ?position2.}\}$$

We assume that the threshold function for $\mathfrak{L}$(?speed) is as: $\tau(n, n') = \text{true}$ if $n' > n$ and $\tau(n, n') = \text{true}$ otherwise. Note the likelihood constraint with threshold function τ on the variable ?speed makes sure that the value of ?speed increases as we move over time. This way we capture the notion of acceleration for the car. Further, the above query is of the form $\Lambda.\Gamma\mathbf{U}\Lambda'$ with $\Lambda = \emptyset$. Further, $\mathfrak{R}_E$(?car) constraints the existence of a car over all time points unless there is a time point T where we we find a second car with position of the first car overlapping that of the second one in the knowledge graph $\mathcal{G} \cup \mathcal{D}_T$.

6. Related Work

Incorporating time into RDF has been investigated in prior work [6, 7, 8, 9, 10, 11]. By and large, these studies differ in aspects like treatment of the notion of time and expressivity of the query language. Contributions like [6] extend the idea of TSQL2 [12] in traditional databases to RDF. The treatment of time there is different from the usual notion of time in classical temporal logic like LTL [13]. Nevertheless, works like [11] extends SPARQL with constructs provided in LTL. A thorough comparison of our approach to the state-of-the-art is beyond the scope of this paper. However, the treatment of time in our work exhibits some similarities to the one presented in [6]. What distinguishes TQL is that it considers only one dimension for time, as our definition of temporal KG is based on a linear order $(\mathbf{T}, <)$ unlike N-dimensional time $\mathcal{T} = \mathcal{T}_1 \times \mathcal{T}_2 \times \cdots \times \mathcal{T}_n$ in [6, 7], where the focus is ontology versioning. When it comes to our query language, our work is somewhat similar to SPARQL-LTL presented in [11]. However, the focus in [11] when it comes to the notion of time is more on versioning of RDF data. Hence, the defined semantics consider different named graphs where each graph represents one version of the data. As pointed out by the authors, it is not a limitation per se. We believe this is comparable to our definition of temporal knowledge graph (Definition 5.2), where without loss of generality, we assume a set of triples to be associated with a time point.

Besides all similarities to the existing approaches, the notion of temporal constraint, to the best of our knowledge, adds new expressivity to SPARQL extended with the temporal operators. Even though such constraints are translated into SPARQL checks, the succinctness they add to the query language is of great advantage, as it helps to reduce complex queries to simpler and more compact ones (as shown in the examples in the traffic monitoring domain).

7. Conclusion and Future Work

We have discussed the benefit of temporal queries in knowledge graphs. We then presented a formal query language for formalizing temporal queries. We called it as Temporal Query Language (TQL). We have precisely defined syntax and semantics of TQL, where we considered different types of temporal queries. We have seen how temporal constraints adds further expressivity to the query language. We have already a first implementation of a query engine for answering temporal queries. A description of the engine, along with a report on experiments and evaluation conducted, is beyond the scope of this paper. In future work, we would like to analyze what other types of temporal queries can be of interest for different use-cases and, in parallel, developing an advanced query editor for TQL.

References

[1] C. Lutz, F. Wolter, M. Zakharyaschev, Temporal description logics: A survey, in: Proceedings of the Fifteenth International Symposium on Temporal Representation and Reasoning, IEEE Computer Society Press, 2008.

[2] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneijder, L. A. Stein, Technical Report, 2004.

[3] F. Baader, I. Horrocks, C. Lutz, U. Sattler, An Introduction to Description Logic, Cambridge University Press, 2017. doi:10.1017/9781139025355.

[4] P. Hitzler, M. Krtzsch, S. Rudolph, Foundations of Semantic Web Technologies, 1st ed., Chapman and Hall/CRC, 2009.

[5] C. Lutz, F. Wolter, M. Zakharyaschev, Temporal description logics: A survey, in: Proceedings of the Fifteenth International Symposium on Temporal Representation and Reasoning, 2008.

[6] F. Grandi, T-SPARQL: A tsql2-like temporal query language for RDF, in: Local Proceedings of the Fourteenth East-European Conference on Advances in Databases and Information Systems, Novi Sad, Serbia, September 20-24, 2010, CEUR Workshop Proceedings, 2010.

[7] F. Grandi, Multi-temporal RDF ontology versioning, in: Proceedings of the 3rd International Workshop on Ontology Dynamics, (IWOD 2009) , collocated with the 8th International Semantic Web Conference (ISWC-2009), Washington DC, USA, October 26, 2009, CEUR Workshop Proceedings, 2009.

[8] J. Tappolet, A. Bernstein, Applied temporal RDF: efficient temporal querying of RDF data with SPARQL, in: The Semantic Web: Research and Applications, 6th European Semantic Web Conference, ESWC 2009, Heraklion, Crete, Greece, May 31-June 4, 2009, Proceedings, 2009.

[9] A. Pugliese, O. Udrea, V. S. Subrahmanian, Scaling RDF with time, in: Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008, 2008.

[10] C. Gutierrez, C. A. Hurtado, A. A. Vaisman, Introducing time into RDF, IEEE Trans. Knowl. Data Eng. (2007).

[11] M. W. Chekol, V. Fionda, G. Pirrò, Time travel queries in RDF archives, in: Joint proceedings

of the 3rd Workshop on Managing the Evolution and Preservation of the Data Web (MEPDaW 2017) and the 4th Workshop on Linked Data Quality (LDQ 2017) co-located with 14th European Semantic Web Conference (ESWC 2017), Portorož, Slovenia, May 28th-29th, 2017, CEUR-WS.org, 2017.

[12] J. Clifford, C. E. Dyreson, R. T. Snodgrass, T. Isakowitz, C. S. Jensen, "now", in: R. T. Snodgrass (Ed.), The TSQL2 Temporal Query Language, Kluwer, 1995, pp. 383–392.

[13] F. Kröger, S. Merz, Basic Propositional Linear Temporal Logic, 2008.