Web Semantics: Science, Services and Agents on the World Wide Web 33 (2015) 50–70

Contents lists available at ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

Temporalizing rewritable query languages over knowledge bases

Stefan Borgwardt, Marcel Lippmann *, Veronika Thost

Institute of Theoretical Computer Science, Technische Universität Dresden, 01062 Dresden, Germany

ARTICLE INFO

Article history:
Received 30 September 2013
Received in revised form
18 November 2014
Accepted 27 November 2014
Available online 8 December 2014

Keywords:
Ontology-based data access
Linear temporal logic
Query answering
Rewritability
Description logic

ABSTRACT

Ontology-based data access (OBDA) generalizes query answering in relational databases. It allows to query a database by using the language of an ontology, abstracting from the actual relations of the database. OBDA can sometimes be realized by compiling the information of the ontology into the query and the database. The resulting query is then answered using classical database techniques.

In this paper, we consider a temporal version of OBDA. We propose a generic temporal query language that combines linear temporal logic with queries over ontologies. This language is well-suited for expressing temporal properties of dynamic systems and is useful in context-aware applications that need to detect specific situations. We show that, if atemporal queries are rewritable in the sense described above, then the corresponding temporal queries are also rewritable such that we can answer them over a temporal database. We present three approaches to answering the resulting queries.

1. Introduction

Context-aware applications try to detect specific situations within a changing environment (e.g., a computer system or air traffic observed by radar) to be able to react accordingly. To gain information, the environment is observed by sensors (for a computer system, data about its resources is gathered by the operating system), and the results of sensing are stored in a database. A context-aware application then detects specific predefined situations based on this data (e.g., a high system load) and reacts accordingly (e.g., by increasing the CPU frequency).

In a simple setting, such an application can be realized by using standard database techniques: the sensor information is stored in a database, and the situations to be recognized are specified as database queries [1]. However, we cannot assume that the sensors provide a complete description of the current state of the environment. Thus, the closed world assumption employed by database systems (i.e., facts not present in the database are assumed to be false) is not appropriate since there may be facts of which the truth is not known. For example, a sensor for specific information might not be available for a moment or not even exist.

In addition, though a complete specification of the environment usually does not exist, some knowledge about its behavior is often available. This background knowledge could be used to formulate constraints on the behavior of the real environment. These constraints help formulate queries to detect more complex situations.

This information (i.e., the sensor data and the background knowledge) is stored in so-called *knowledge bases*, which are sometimes called *ontologies*. A knowledge base consists of a fact base and a theory, which store the data in a formally well-understood way. The *fact base* contains simple facts (e.g., the concrete values given by sensors), and is interpreted with the *open world assumption*, i.e., facts not present are assumed to be unknown rather than false. The *theory* contains the additional background knowledge (e.g., general domain knowledge) stored in a symbolic representation. The situations to be detected are then specified in an appropriate query language. The resulting queries are then evaluated w.r.t. the information encoded in the knowledge base. This general approach is often called ontology-based data access (OBDA) [2,3].

However, since the environment is changing, it is often desirable to specify situations that take into account *temporal* behavior. In this setting, we model the incoming information as a *sequence* of fact bases, one for each moment in time in which the system has been observed. To recognize situations, we propose to add a temporal logical component to atemporal queries over knowledge bases. We use the operators of the temporal logic LTL, which allows to reason about a linear and discrete flow of time [4]. Usual temporal operators include *next* ($\bigcirc\phi$), which asserts that a property ϕ is true at the next point in time, *eventually* ($\Diamond\phi$), which requires ϕ to be satisfied at some point in the future, and *always* ($\Box\phi$), which

* Corresponding author.
E-mail addresses: stefborg@tcs.inf.tu-dresden.de (S. Borgwardt), lippmann@tcs.inf.tu-dresden.de (M. Lippmann), thost@tcs.inf.tu-dresden.de (V. Thost).

http://dx.doi.org/10.1016/j.websem.2014.11.007
1570-8268/

forces ϕ to be true at all time points in the future. We also use the corresponding past operators $\bigcirc^-$, $\Diamond^-$, and $\Box^-$.

Consider, for example, a distributed video platform providing several services such as uploading, streaming, and transcoding (i.e., the conversion of video formats). At any given time point, a fact base for such a system could contain facts like the following, which describe that there is a server s with an overutilized CPU c, which executes an uploading service (ULS) p_1 and a transcoding service (TCS) p_2, both of which are active:

$$\begin{array}{llll} \mathsf{CPU}(c), & \mathsf{Overutilized}(c), & \mathsf{Server}(s), & \mathsf{hasCPU}(s,c), \\ \mathsf{ULS}(p_1), & \mathsf{executes}(s,p_1), & \mathsf{Active}(p_1), & \\ \mathsf{TCS}(p_2), & \mathsf{executes}(s,p_2), & \mathsf{Active}(p_2). & \end{array}$$

The background theory could contain an axiom such as

$$\forall x.\mathsf{Server}(x) \wedge \big(\exists y.\mathsf{hasCPU}(x,y) \wedge \mathsf{Overutilized}(y)\big) \rightarrow \mathsf{Overloaded}(x),$$

which states that a server having an overutilized CPU is overloaded. Given the above fact base, we can conclude that s is currently overloaded.

Since transcoding is very resource-intensive, it is important to transcode popular videos preemptively in phases of less utilization instead of on demand in phases of high utilization. However, the situation can clearly change after a preemptive transcoding service has been started. For that reason, one may want to detect critical situations in which a server of the platform has become overloaded while executing such a service.

The temporal query

$$\mathsf{TCS}(x) \wedge \mathsf{Server}(y) \wedge \mathsf{executes}(y,x) \wedge \psi_0 \wedge \big(\mathsf{NLB}(y)\,\mathsf{S}\,\bigcirc\psi_t\big)$$

with

$$\psi_t := \begin{cases} \mathsf{Active}(x) \wedge \mathsf{Overloaded}(y) & \text{if } t = 0 \\ \psi_0 \wedge \bigcirc\Diamond\psi_{t-1} & \text{if } t \geq 1 \end{cases}$$

and $t \geq 0$ therefore asks for a transcoding service x and a server y that executes it, where x is active and y is overloaded. The second part of the query requires that $\mathsf{NLB}(y)$ has been true for the whole time *since* (S) the subquery $\bigcirc\psi_t$ was true. In other words, we are looking for a time point in the past that satisfies ψ_t such that all time points since then satisfy $\mathsf{NLB}(y)$, which expresses that y has not been affected by a load balancing operation in the meantime. The subquery ψ_t again asks for x to be active and y to be overloaded, and furthermore that there is a time point after the current one ($\bigcirc\Diamond$) satisfying ψ_{t-1}. We are thus asking for a series of $t+1$ critical time points (not necessarily immediately following each other). We consider the temporal behavior of this example query in more detail in Sections 5 and 6.

One might argue that, as we are looking at the time line from the point of view of the current time point, and nothing is known about the future, it is sufficient to have only past operators like S or $\Box^-$. We also show that in our setting it is indeed always possible to construct an equivalent query using only past operators (see Section 5.3). However, the resulting query is not very concise and it is not easy to see the situation that is to be recognized. Indeed, for propositional LTL eliminating the *past operators* from a formula results in a blowup that is at least exponential and no constructions of size less than triply exponential are known [5].

1.1. Related work

In this paper, we consider so-called *rewritable query languages*, i.e., query languages for which evaluating a query over a knowledge base can be reduced to answering a rewritten query (in a different language) over a database induced by the knowledge base. Such query languages, especially in the context of Description Logics (DLs) [6], are covered extensively in the literature (see Example 2.11).

Investigations of *temporal* query languages based on combinations of query languages and temporal logics such as LTL [4] have started only quite recently. Yet, a number of very expressive temporal query languages have been proposed [7–10].

For rewritable query languages, most research focuses on lightweight languages of the *DL-Lite* family [11]. However, instead of temporalizing the query language and evaluating the queries over a global knowledge base, also temporal knowledge bases are examined, which allow temporal operators to occur inside axioms. These approaches are based on research about temporalized description logics (see [12] for a survey). For example, in [13], various lightweight DLs are extended by allowing the temporal operators to interfere with the DL component. Following the ideas of [13], in [14] a rewritable temporal query language over temporal knowledge bases in *DL-Lite* is proposed.

There is also a lot of closely related work in the field of temporal databases. In [15], for instance, the authors describe a temporal extension of the SQL query language that can answer temporal queries over a temporal database. In [16–18], an approach is described that reduces the amount of space needed to evaluate temporal queries by keeping only the relevant data in the database instead of keeping track of all the information from the past.

1.2. Our contribution

In this paper, we consider temporal queries over knowledge bases in a very general setting that allows us to extend many existing atemporal query languages by temporal operators (cf. Section 3). In Section 4, we show that the reasoning task of *temporal OBDA* in this setting can be reduced to answering queries over temporal databases. The main part of the paper is thus concerned with what we call the *temporal database monitoring problem*, where a fixed temporal query is continuously evaluated over a temporal sequence of databases.

We present three approaches to solving this problem. The first one employs existing temporal database systems using a translation from our temporal query language into a specialized database query language [15] (cf. Section 5.1). The second approach again rewrites the query in order to obtain a query without future operators, which then can be answered using an algorithm from [16] (cf. Section 5.3). The advantage of this algorithm is that the time required to answer the temporal query at the current time point does not depend on the total running time of the system; this is called a *bounded history encoding* in [16]. In Section 6, we propose a new algorithm that extends the one from [16] in that it also deals with future operators directly while guaranteeing a bounded history encoding. We also discuss different advantages and drawbacks of the three approaches.

Sometimes it is desired to state that certain facts do not change over time, i.e., are *rigid*. In Section 7, we show how our proposed algorithm can be extended to deal with a limited form of rigidity in a specific class of queries.

This paper is an extension of [19], where we have considered only the special case of answering temporal queries over $\mathit{DL\text{-}Lite}_{core}$-ontologies. In contrast to [19], we also show in this paper that our proposed algorithm preserves the bounded history encoding of [16]. Additionally, this paper contains the full proofs of our results. To improve readability, some of them are presented in the Appendix.

2. Preliminaries

As mentioned in the introduction, we consider temporal queries over knowledge bases in a very general setting. This section