Dresden University of Technology
Institute for Theoretical Computer Science
Chair for Automata Theory

LTCS–Report

On Implementing Temporal Query Answering in *DL-Lite*

Veronika Thost Jan Holste Özgür Özçep

LTCS-Report 15-12

Postal Address:
Lehrstuhl fr Automatentheorie
Institut fr Theoretische Informatik
TU Dresden
01062 Dresden

http://lat.inf.tu-dresden.de

Visiting Address:
Nthnitzer Str. 46
Dresden

On Implementing Temporal Query Answering in *DL-Lite*

Veronika Thost[1] Jan Holste[2]
Özgür Özçep[3]

[1]Technische Universität Dresden, Germany,
thost@tcs.inf.tu-dresden.de
[2]Technische Universität Hamburg-Harburg, Germany,
mail@janholste.com
[3]Universität zu Lübeck, Germany,
oezcep@ifis.uni-luebeck.de

Abstract

Ontology-based data access augments classical query answering over fact bases by adopting the open-world assumption and by including domain knowledge provided by an ontology. We implemented temporal query answering w.r.t. ontologies formulated in the Description Logic *DL-Lite*. Focusing on temporal conjunctive queries (TCQs), which combine conjunctive queries via the operators of propositional linear temporal logic, we regard three approaches for answering them: an iterative algorithm that considers all data available; a window-based algorithm; and a rewriting approach, which translates the TCQs to be answered into SQL queries. Since the relevant ontological knowledge is already encoded into the latter queries, they can be answered by a standard database system. Our evaluation especially shows that implementations of both the iterative and the window-based algorithm answer TCQs within a few milliseconds, and that the former achieves a constant performance, even if data is growing over time.

1 Introduction

Temporal information plays a central role in many applications of ontology-based data access (OBDA). For example, knowledge about the past is usually kept in patient records, and collected by companies or scientific projects such as MesoWest[1], focusing

[1] http://mesowest.utah.edu/

on weather data. Such applications obviously benefit of using ontologies for data integration and access. For example, the wind force 'Storm' on the well-known Beaufort Wind Force Scale is equally characterized by wind speed and wave height. This can be represented in an ontology by an expression such as $\mathsf{HighWindSpeed} \sqcup \mathsf{HighWaves} \sqsubseteq \mathsf{Storm}$, saying that if either a 'high wind speed' or 'high waves' are observed, this implies the wind force 'Storm'. Regarding such an ontology, the conjunctive query (CQ) $\mathsf{Storm}(x)$ over the fact base $\{\mathsf{HighWindSpeed}(NYC)\}$ would then retrieve the tuple (NYC) as answer. Temporal knowledge is however not taken into account by systems implementing OBDA, in general. Though, assuming that we consider several weather stations' data of the past 24 hours, a query such as the following could be interesting: *"Get the heritage sites that are nearby a weather station, for which at some time in the past (24 hours) a danger of a hurricane was detected, since then, the wind force has been continuously very high, and it increased considerably during the two latest times of observation."*

For that reason, we focus on the practical answering of *temporal conjunctive queries* (TCQs) [4, 5] w.r.t. ontologies written in the description logic (DL) *DL-Lite*$_{core}$ (hereinafter called *DL-Lite*). TCQs combine conjunctive queries via LTL operators[2] and have been already studied extensively in the context of *DL-Lite* [17, 8]. The above example query could be specified as the following TCQ:

$$\mathsf{HeritageSite}(x) \wedge \mathsf{WeatherStation}(y) \wedge \mathsf{nearby}(x, y) \wedge$$
$$\big(\mathsf{HighWind}(y)\ \mathsf{S}\ \mathsf{DangerOfHurricane}(y)\big) \wedge\ \bigcirc^{-}\ \mathsf{Storm}(y) \wedge \mathsf{ViolentStorm}(y),$$

asking for all pairs (x, y) of heritage sites and nearby weather stations, whose sensor values at some point in time implied a danger of a hurricane, *since* (S) then, the measurements have implied Beaufort category 'high wind', in the *previous* ($\bigcirc^{-}$) moment of observation they implied category 'storm', and the latest values imply 'violent storm'. TCQs are answered w.r.t. *temporal knowledge bases* (TKBs), which, in addition to the ontology (assumed to hold at every point in time), contain a sequence of fact bases $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_n$, representing the data collected at specific points in time. Especially note that the ontology and the fact bases itself are formulated in a classical (i.e., atemporal) DL, only the queries are temporal.

Related Work

On the DL side, there are various optimized systems realizing OBDA [14]. In particular, the so-called *rewriting approach* implemented by Ontop [19] allows for efficient query answering w.r.t. an ontology written in the DL *DL-Lite*. Specifically, Ontop internally rewrites a given conjunctive query, which is written in the abstract vocabulary of the ontology, into an SQL query that encodes the relevant ontological knowledge but addresses a standard database system; the latter can then be used to store the data and efficiently answer the queries. However, whereas a lot of DL

[2]Please note that we do not consider negation.

research is theoretically investigating temporal extensions of ontology and query languages [2, 6, 3, 15, 17, 5, 8, 18], none of the freely available systems takes the temporal nature of the data into account, yet (i.e., the query languages supported do not provide operators for explicitly referencing different points in time).

Recently, several practical approaches for answering temporal queries have been developed in the fields of stream reasoning [7] and complex event processing [13, 1]. These systems are tailored to continuously answering given queries over an infinite stream of data—which is usually realized by restricting the focus to a so-called *window* of the data (i.e., instead of considering all the past data available, the number of considered time points or data instances is fix). The proposed query languages are rather expressive, but ontologies are not yet integrated, in general; only Morph-streams [12] presents an exception. Similar to Ontop, Morph-streams provides OBDA to streaming data sources by rewriting the given queries to queries that can be answered by existing stream processors; it therefore supports a query language that extends CQs, which is tailored to the ontology and, at the same time, window-based querying of data streams. Nevertheless, common standards for stream representation and processing, for query languages, and for operation semantics are still to be developed in these fields.

Our work complements these applied approaches by starting from a DL perspective, where many use cases consider static data, all of which might be relevant; ontologies are important; and there are several well-investigated query languages. Specifically, we study three pragmatic approaches for answering TCQs based on the work of [8]. We prototypically implemented and evaluated them, and in this paper report on our experiences.

Algorithms for Temporal Query Answering

In particular, [8] consider a scenario in which the sequence of fact bases is continuously extended and a fix set of TCQs is answered with every extension, at time point n—similar to the streaming scenario.

We first implemented the Iterative Algorithm (IA) (cf. Section 6 in [8]), which iteratively computes sets of answers to several subqueries of the TCQ to be answered, for each time point i, $0 \leq i \leq n$. For example, the answers to $\bigcirc^{-}\mathsf{Storm}(x)$ at i are obtained by evaluating $\mathsf{Storm}(x)$ at $i-1$. Since the processing at i only uses $\mathcal{A}_i$ and the sets computed for the previous moment, whose sizes are bounded, the IA achieves a so-called *bounded history encoding* (i.e., it's runtime does not depend on the number of considered fact bases). The latter is however due to the constant domain assumption of [8], which might not fit in an application (e.g., in a streaming scenario based on social network data). We therefore describe a window-based variant of the IA, the Vector Algorithm (VA) (cf. Section 4 in [16]). The VA specifically supports *sliding* windows, where the TCQs are not evaluated at every time point, but in fix intervals. It therefore regards a sequence (or *vector*) containing only the then necessary of the

above mentioned sets of answers. To answer the query $\bigcirc^{-}\mathsf{Storm}(x)$ at every second point in time, for example, $\mathsf{Storm}(x)$ does not always have to be evaluated. In order to get an impression of the performance of a temporal rewriting approach targeting data stored on disk, we finally consider the (rather simple) Rewriting Algorithm (RA) translating TCQs into standard database queries.

Our evaluation shows that the implementations of both the IA and VA perform surprisingly well: TCQs w.r.t. TKBs containing millions of facts are answered within milliseconds. We could however not obtain such results for the RA, mainly due to the simplicity of the rewriting and because the disk access was too costly. In what follows, we first give basic knowledge about *DL-Lite* and TCQs, then describe the algorithms, and finally detail our evaluation results.

2 Preliminaries

We briefly introduce the description logic *DL-Lite*, especially as ontology language, and TCQs. Description Logics generally distinguish the terminological knowledge, which is collected in an *ontology*, and the facts about a particular environment, collected in a *fact base*. In particular, DLs model the domain of interest using *concepts*, denoting classes of individuals, and *roles*, representing binary relations between individuals. As in other logics, expressions in DLs are built over an alphabet of named symbols. We discern *individual names*, which represent constants, *concept names*, and *role names*.

Definition 2.1 (Syntax of *DL-Lite*) *In DL-Lite$_{core}$,* basic concepts B, (general) concepts C, *and* basic roles R *are built from concept names* A *and role names* P *according to the following syntax rules:*

$$B ::= A \mid \exists R \qquad\qquad R ::= P \mid P^{-}$$
$$C ::= B \mid \neg B$$

where $\cdot^{-}$ *denotes the inverse-role operator.*

Let now A *be a concept name,* B *be a basic concept,* C *be a general concept, and* P *be a role name. We focus on two types of expressions:* concept inclusions (CIs)*, of the form* $B \sqsubseteq C$*, and* assertions*, of the form* $A(a)$ *or* $P(a, b)$*, where* a *and* b *denote individual names. Further, an* ontology *is a finite set of concept inclusions,*[3] *and a* fact base *is a finite set of assertions.*

In the context of the rewriting approach, the data store is assumed to be the fact base containing the assertional knowledge. This can be implemented in a database, for

[3]Note that, in literature, the notion ontology usually includes role inclusions, expressions that are allowed in DLs that are more expressive than *DL-Lite$_{core}$*. That is, the ontology then comprises both the set of CIs (called TBox) and the set of role inclusions (called RBox).

example, by designing it such that there is a table for every concept name and role name [9], and especially shows the advantage of this approach—namely, that the data does not have to be converted into a specific format, any more, but can be used as it is usually given, stored in a conventional data management system.

We define the semantics of *DL-Lite* as usual in DLs, in terms of First-Order Logic interpretations. Further, we require all these interpretations to agree on the semantics assigned to the constants, the individual names. This means that each such name is assigned to a unique value of the domain, in every interpretation (also known as *unique name assumption*). It also implies that different individual names are interpreted differently in every domain.

Definition 2.2 (Semantics of *DL-Lite*) *An* interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *is a pair consisting of a non-empty set* $\Delta^{\mathcal{I}}$ *(called* domain*) and an* interpretation function $\cdot^{\mathcal{I}}$ *that assigns to every concept name* A *a set* $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$*, to every role name* P *a binary relation* $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$*, and to every individual name* a *an element* $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$*. This function is extended to all roles and concepts as follows:*

$$(R^-)^{\mathcal{I}} := \{(y, x) \mid (x, y) \in R^{\mathcal{I}}\};$$
$$(\exists R)^{\mathcal{I}} := \{x \mid \text{there is an } y \in \Delta^{\mathcal{I}} \text{ such that } (x, y) \in R^{\mathcal{I}}\}; \text{ and}$$
$$(\neg C)^{\mathcal{I}} := \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}.$$

The interpretation $\mathcal{I}$ satisfies

- *a CI* $B \sqsubseteq C$ *if* $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$*;*
- *an assertion* $A(a)$ *if* $a^{\mathcal{I}} \in A^{\mathcal{I}}$*;*
- *an assertion* $P(a, b)$ *if* $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in P^{\mathcal{I}}$*;*
- *an ontology* $\mathcal{O}$ *if it satisfies all CIs contained in it (written* $\mathcal{I} \models \mathcal{O}$*);*
- *a fact base* $\mathcal{A}$ *if it satisfies all assertions contained in it (written* $\mathcal{I} \models \mathcal{A}$*).*

Focusing on temporal OBDA, we especially do not restrict ourselves to the knowledge known about a single moment in time. That is, we do not only consider a single fact base, but a *sequence of fact bases* $\mathcal{A}_0, \ldots, \mathcal{A}_n$ and assume each of the fact bases to contain knowledge about a particular point in time; thus, n ('now') denotes the latest moment for which data is available. In contrast, we assume the ontological knowledge to be always true (i.e., independent of the time point considered).

Definition 2.3 (Temporal Knowledge Base) *A* temporal knowledge base *(TKB)* $\mathcal{K} = \langle \mathcal{O}, (\mathcal{A}_i)_{0 \leq i \leq n} \rangle$ *consists of an ontology* $\mathcal{O}$ *and a finite sequence of fact bases* $\mathcal{A}_i$*.*

Let $\mathfrak{I} = (\mathcal{I}_i)_{i \geq 0}$ *be an infinite sequence of interpretations* $\mathcal{I}_i = (\Delta, \cdot^{\mathcal{I}_i})$ *over a non-empty domain* Δ *that is fixed (constant domain assumption). Then,* $\mathfrak{I}$ *is a* model *of* $\mathcal{K}$ *(written* $\mathfrak{I} \models \mathcal{K}$*) if*

- *for all $i \geq 0$, we have $\mathcal{I}_i \models \mathcal{O}$; and*
- *for all i, $0 \leq i \leq n$, we have $\mathcal{I}_i \models \mathcal{A}_i$.*

We finally describe our query language, which combines conjunctive queries via LTL operators.

Definition 2.4 (Syntax of TCQs) *A* conjunctive query *(CQ) is of the form $\psi(\vec{x}) = \exists \vec{y}.\psi'(\vec{x}, \vec{y})$, where $\vec{x}$ and $\vec{y}$ are tuples of (pairwise distinct) variables, and $\psi'(\vec{x}, \vec{y})$ is a (possibly empty) finite conjunction of atoms of the form*

- $A(t)$ *(*concept atom*), for a concept name A, or*
- $P(t, u)$ *(*role atom*), for a role name P,*

where t, u are either individual names or variables that occur in ψ. The empty conjunction is denoted by true*. The variables in $\vec{x}$ are called* distinguished *variables (vs.* undistinguished*).*

Temporal conjunctive queries *(TCQs) are built from CQs as follows:*

- *Every CQ is a TCQ.*
- *If ϕ_1 and ϕ_2 are TCQs, then the following are also TCQs:*
  - *$\phi_1 \wedge \phi_2$ (conjunction), $\phi_1 \vee \phi_2$ (disjunction),*
  - *$\bigcirc \phi_1$ (next), $\bigcirc^- \phi_1$ (previous),*
  - *$\phi_1 \,\mathsf{U}\, \phi_2$ (until), $\phi_1 \,\mathsf{S}\, \phi_2$(since),*
  - *$\Box \phi_1$ (always), and $\Box^- \phi_1$ (always in the past).*

As usual, we use the abbreviation $\Diamond \phi_1$ (eventually) for $\mathsf{true}\,\mathsf{U}\,\phi_1$, and analogously for the past, $\Diamond^- \phi_1$, for $\mathsf{true}\,\mathsf{S}\,\phi_1$.

Given an interpretation, we next describe the *answers* to TCQs in that interpretation as mappings from the variables and individual names occurring in them to elements of the domain of the interpretation. Especially, we require the distinguished variables to be mapped to domain elements that represent individual names of the knowledge base. We start by defining the semantics of CQs for Boolean queries. As usual, it is given through the notion of homomorphisms [11].

Definition 2.5 (Semantics of TCQs) *Let $\psi(x_1, \ldots, x_m)$ be a CQ, $(a_1, \ldots, a_m)$ be a tuple of individual names of same arity as the tuple $(x_1, \ldots, x_m)$, and $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation. A mapping π from the variables and individual names that occur in ψ to the elements of $\Delta^{\mathcal{I}}$ is a homomorphism of $\psi(a_1, \ldots, a_m)$ into $\mathcal{I}$ if*

ϕ	$\mathfrak{I}, i \models \mathfrak{a}(\phi)$ iff
A CQ ψ	$\mathcal{I}_i \models \mathfrak{a}(\psi)$
$\phi_1 \wedge \phi_2$	$\mathfrak{I}, i \models \mathfrak{a}_{\phi_1}(\phi_1)$ and $\mathfrak{I}, i \models \mathfrak{a}_{\phi_2}(\phi_2)$
$\phi_1 \vee \phi_2$	$\mathfrak{I}, i \models \mathfrak{a}_{\phi_1}(\phi_1)$ or $\mathfrak{I}, i \models \mathfrak{a}_{\phi_2}(\phi_2)$
$\bigcirc \phi_1$	$i < n$ and $\mathfrak{I}, i+1 \models \mathfrak{a}(\phi_1)$
$\bigcirc^- \phi_1$	$i > 0$ and $\mathfrak{I}, i-1 \models \mathfrak{a}(\phi_1)$
$\Box \phi_1$	$\mathfrak{I}, k \models \mathfrak{a}(\phi_1)$ for all k, $i \leq k \leq n$
$\Box^- \phi_1$	$\mathfrak{I}, k \models \mathfrak{a}(\phi_1)$ for all k, $0 \leq k \leq i$
$\phi_1 \mathsf{U} \phi_2$	there is k, $i \leq k \leq n$, with $\mathfrak{I}, k \models \mathfrak{a}_{\phi_2}(\phi_2)$ and $\mathfrak{I}, j \models \mathfrak{a}_{\phi_1}(\phi_1)$ for all j, $i \leq j < k$
$\phi_1 \mathsf{S} \phi_2$	there is k, $0 \leq k \leq i$, with $\mathfrak{I}, k \models \mathfrak{a}_{\phi_2}(\phi_2)$ and $\mathfrak{I}, j \models \mathfrak{a}_{\phi_1}(\phi_1)$ for all j, $k < j \leq i$

Table 1: The semantics of TCQs.

- $\pi(x_i) = \pi(a_i)$, *for all* $1 \leq i \leq m$;
- $\pi(a) = a^{\mathcal{I}}$, *for all individual names* a *occurring in* ψ;
- $\pi(t) \in A^{\mathcal{I}}$, *for all concept atoms* $A(t)$ *in* ψ; *and*
- $(\pi(t), \pi(u)) \in P^{\mathcal{I}}$, *for all role atoms* $P(t, u)$ *in* ψ.

We say that $(a_1, \ldots, a_m)$ *is an* answer *to* ψ *in* $\mathcal{I}$ *(written* $\mathcal{I} \models \psi(a_1, \ldots, a_m)$*) iff there is such a homomorphism.*

Let now ϕ *be a TCQ,* $\mathfrak{a}$ *be a mapping from the distinguished variables in* ϕ *to individual names,* i *be an integer with* $0 \leq i \leq n$*, and* $\mathfrak{I} = (\mathcal{I}_i)_{i \geq 0}$ *be an infinite sequence of interpretations. We define the satisfaction relation* $\mathfrak{I}, i \models \phi$ *by induction on the structure of* ϕ *as described in Table 2.5; and, if* $\mathfrak{I}, i \models \mathfrak{a}(\phi)$*, then we call* $\mathfrak{a}$ *an* answer *to* ϕ *w.r.t.* $\mathfrak{I}$ *at time point* i.

We finally come to the problem this paper is about and which is targeted by our algorithms. In particular, we focus on *certain answers*, which are answers to the given TCQ, in *every* model of the considered TKB.

Definition 2.6 (Certain Answer) *Let* ϕ *be a TCQ and* $\mathcal{K} = \langle \mathcal{O}, (\mathcal{A}_i)_{0 \leq i \leq n} \rangle$*, be a TKB. The mapping* $\mathfrak{a}$ *from the distinguished variables of* ϕ *to individual names that occur in* $\mathcal{K}$ *is a* certain answer *to* ϕ *w.r.t.* $\mathcal{K}$ *at time point* i *if, for all models* $\mathfrak{I}$ *of* $\mathcal{K}$*, we have* $\mathfrak{I}, i \models \mathfrak{a}(\phi)$.

We denote the set of certain answers to a TCQ ϕ w.r.t. a TKB $\mathcal{K} = \langle \mathcal{O}, (\mathcal{A}_i)_{0 \leq i \leq n} \rangle$ at i by $\mathsf{Cert}(\phi, \mathcal{K}, i)$. We subsequently describe three approaches to solve the problem of finding the certain answers to TCQs in a pragmatic way.

3 The Iterative Algorithm

We start by describing the idea of the Iterative Algorithm. Please consider the original definition in [8, Section 6] for the detailed definition. We assume ϕ to be the TCQ answered w.r.t. a TKB $\mathcal{K}$. In a nutshell, the Iterative Algorithm computes a set $\Phi_i(\phi_0)$ of answers to ϕ_0 (w.r.t. $\mathcal{K}$) at i, for several subqueries ϕ_0 of ϕ and all time points i, $0 \leq i \leq n$. As the name suggests, it thereby proceeds iteratively, starting in the initial time point 0 (e.g., the set $\Phi_i(\bigcirc^- \phi_1)$ of answers to a TCQ $\bigcirc^- \phi_1$ at time point i, is defined as $\Phi_i(\bigcirc^- \phi_1) := \Phi_{i-1}(\phi_1)$, based on the set $\Phi_{i-1}(\phi_1)$ of answers to ϕ_1 computed for time point $i-1$, if $i > 0$ and otherwise as the empty set). It thereby always proceeds along the part-of order of the queries reusing sets it computed before. Further, CQs can be evaluated over the data of the current step, $\mathcal{A}_i$; that is, for a given CQ ψ and TKB $\mathcal{K}$, we have $\Phi_i(\psi) := \mathsf{Cert}(\psi, \mathcal{K}, i)$. In this way, the Boolean operators and the $\bigcirc^-$ operator can be resolved in accordance with the semantics (e.g., the $\wedge$ operator can be resolved by intersecting the sets of answers computed for the two subqueries). However, since the subqueries may refer to knowledge of future time points, Φ_i actually does not always map to a set of *answers* as defined in Section 2. More specifically, it refers to an expression over sets of answers that may contain placeholder variables of the form $x_i^{\bigcirc \phi_2}$ or $x_i^{\phi_3 \,\mathsf{U}\, \phi_4}$. $x_i^{\bigcirc \phi_2}$ represents $\Phi_i(\bigcirc \phi_2)$, the answers to the query $\bigcirc \phi_2$ at i, because the query directly refers to future time point(s). Similarly, the answers to $\phi_3 \,\mathsf{U}\, \phi_4$ at i, are given by $\Phi_i(\phi_3 \,\mathsf{U}\, \phi_4) := \Phi_i(\phi_4) \cup (\Phi_i(\phi_3) \cap x_i^{\phi_3 \,\mathsf{U}\, \phi_4})$. We therefore refer to *answer terms* instead of 'sets of answers', in the following (e.g., for a query $\phi := (\bigcirc^- \phi_1) \wedge \bigcirc \phi_2$ with $i > 0$, we have $\Phi_i(\phi) = \Phi_{i-1}(\phi_1) \cap x_i^{\bigcirc \phi_2}$).

In particular, all the answer terms that need to be considered at some time point i can be computed based on $\mathcal{A}_i$ and the sets of answer terms that were computed for $i-1$. The certain answers to ϕ at time point i are then obtained by adequately evaluating $\Phi_i(\phi)$. During this evaluation, especially the placeholder variables are replaced by either sets of answers (e.g., for $i < n$, $x_i^{\phi_3 \,\mathsf{U}\, \phi_4}$ is replaced by $\Phi_{i+1}(\phi_3 \,\mathsf{U}\, \phi_4)$) or $\emptyset$ if $i = n$, according to the future knowledge available through the (whole) sequence of fact bases given (e.g., for $\phi_1(x) := A(x) \,\mathsf{U}\, B(x)$, $\phi_2(x) := C(x)$, $A, B, C \in \mathsf{N_C}$, and a TKB with empty ontology and the fact bases $\mathcal{A}_0 = \{A(a)\}, \mathcal{A}_1 = \mathcal{A}_n = \{B(a)\}$, we have: $\Phi_1(\phi_1)$ evaluates to $\{(a)\}$ since $\Phi_1(A(x))$ and $\Phi_2(\phi_1(x))$, which replaces $x_1^{A(x) \,\mathsf{U}\, B(x)}$, evaluate to $\{(a)\}$; though, $\Phi_n(\phi)$ evaluates to $\emptyset$ because $x_n^{\bigcirc \phi_2}$ does so).

Note that, because of the constant domain assumption, we have that the sizes of the evaluated answer terms are bounded. The IA thus achieves a so-called *bounded history encoding*. Nevertheless, the consideration of all the historical data might be neither feasible nor required in several practical applications. We therefore propose an adaptation of the IA, next.

4 The Vector Algorithm

The Vector Algorithm generally follows the idea of the IA in that it evaluates answer terms based on the part-of relation of the (sub)queries to be answered. It however specifically supports so-called *sliding windows*. In several applications (e.g., stream processing), the increasing size of the input is managed by applying window operators cutting a *window* of finite size (also *range*) r out of all the available (time-stamped) data. Thereby, the window may either consists of the r most recent data items (for row based windows) or of all the items given for the last r time points (for time based windows). Since we assume a sequence $\mathcal{S}$ of fact bases given, we follow the latter approach. The window cut out at time point $t \geq 0$ is denoted by $\mathcal{S}[t, r, s]$, where the parameter s defines how much and how often the window moves (its *slide*). In that way, the time points for querying can be specified more fine-granularly, and the window only moves at each multiple m of s:

$$\mathcal{S}[t, r, s] = \begin{cases} () & \text{if } t < s - 1; \\ (\mathcal{A}_{t'-j})_{\min(r-1,t') \geq j \geq 0} & \text{else, where } m = (t+1)/s \text{ and } ms = t' + 1. \end{cases}$$

For a given sequence of fact bases $\mathcal{S}$, a time point t, range r, slide s, ontology $\mathcal{O}$, and TCQ ϕ, the Vector Algorithm then calculates the set of certain answers to ϕ w.r.t. $(\mathcal{O}, \mathcal{S}[t, r, s])$ at time point $r - 1$ (i.e., we regard the window as sequence of r fact bases). Obviously, the IA can be used for this computation, too. However, as outlined in Section 1, a specific slide may considerably reduce the sets of answer terms that *have to* be evaluated to answer ϕ w.r.t. $(\mathcal{O}, \mathcal{S}[t, r, s])$ at the last time point of the window. For that reason, the VA does not evaluate all the answer terms as defined in [8]—it might not even be necessary to iterate over all the time points in $\mathcal{S}[t, r, s]$—, but only considers those actually required. To identify these terms, r and s must be known in the beginning of the processing. For a TCQ to be evaluated (at $r - 1$) over a given window, the VA then specifies a *sequence* (or *vector*) of sets of answer terms, which especially do not contain placeholder variables. Using such a vector, we can refer to specific points in the window such that the corresponding answer term does not have to be evaluated for all time points of the window, but still can refer to the whole window, by adding the corresponding answer terms everywhere in the sequence (e.g., for the TCQ $\phi := (\bigcirc^- \phi_0) \wedge \Box^- \phi_1$ with ϕ_0, ϕ_1 being CQs, and a range $r \geq 4$, we obtain the sequence $\{\Phi_0(\phi_1)\}$,…, $\{\Phi_{r-3}(\phi_1)\}$,$\{\Phi_{r-2}(\phi_0), \Phi_{r-2}(\phi_1)\}$,$\{\Phi_{r-1}(\phi_1)\}$). The detailed definition of these vectors as well as the specification of their evaluation, which appropriately combines the answers of the evaluated answer terms (e.g., the certain answers to ϕ are given through $\left(\bigcap_{0 \leq i \leq r-1} \Phi_i(\phi_1)\right) \cap \Phi_{r-2}(\phi_0)$), can be found in [16]. In the latter work, we also describe an optimized version of the VA, which exploits overlapping windows, such that the results of already evaluated answer terms are reused in computations for subsequent windows.

We finally propose a very simple and straightforward rewriting approach to get an impression of how state-of-the-art databases cope with TCQs if they are rewritten into SQL.

5 A Straightforward Rewriting Approach

We now show how a TCQ ϕ to be answered w.r.t. a TKB $\mathcal{K}$ can be rewritten into an SQL query $\mathsf{sql}(\phi)$ such that the answers obtained by answering $\mathsf{sql}(\phi)$ over a database storing the assertions of the fact bases of $\mathcal{K}$ are exactly the certain answers to ϕ w.r.t. $\mathcal{K}$. Note that the idea is similarly described by [8], where it is especially proven that the approach is correct. However, in [8], the TCQ is assumed to be rewritten into a *temporal* standard query language. Since it turned out that such query languages are only rarely supported by existing databases, we propose a rewriting into general SQL.

As described in [9], the rewriting approaches generally use a specific mapping which describes how the contents of the database relate to the concepts of the ontology and, especially, how specific data items relate to assertions of a fact base. This mapping thus specifies the sequence of fact bases represented by the given database. For example, the mapping may describe that the sensor data in the table $\mathsf{datatable}(\mathsf{loc}, \mathsf{sensor}, \mathsf{value}, \mathsf{timestamp})$ is used to instantiate concepts such as HighWindSpeed and HighWaves with individuals representing the location of the corresponding sensors, based on the actual contents of the table (e.g., for a tuple $(NYC, \mathit{wind\text{-}speed}$,14,12:04:09), such a mapping could determine that the fact base associated with the time point '12:04:09' contains an assertion $\mathsf{HighWindSpeed}(NYC)$ based on the value of 14 m/s). For a detailed description of this mapping, we refer to [9].

We assume such a mapping given and further that, for a CQ ψ, the function $\mathsf{sql}(\psi)$ returns the SQL rewriting of ψ as it is proposed in [10, 9]. We specifically assume $\mathsf{sql}(\psi)$ to refer to a column timestamp. Note that this rewriting of CQs already integrates the information specified in the ontology containing concept inclusions such as $\mathsf{HighWindSpeed} \sqcup \mathsf{HighWaves} \sqsubseteq \mathsf{Storm}$.[4] For example, the CQ $\mathsf{Storm}(x)$ could be translated by the function sql as follows:

```
SELECT loc, sensor, timestamp FROM datatable
        WHERE (sensor = 'wind-speed' AND value > 24.5) OR
                (sensor = 'wave-height' AND value > 9);
```

The Rewriting Algorithm uses these SQL queries and thus only needs to specifically address the temporal operators according to the semantics of TCQs. For example, the TCQ $\bigcirc^{-}\mathsf{Storm}(x)$ could be translated into the below SQL query:

```
SELECT loc FROM sql(Storm(loc))
WHERE timestamp = prevTimeStamp(current_time);
```

where the selection condition requires the timestamp to equal the time point before the current moment—which we assume to be given—by using a function prevTimeStamp that determines that previous time point. The translations of all the temporal operators

[4]Note that this kind of expression using disjunction ($\sqcup$) on the left-hand side is not explicitly introduced in Section 2. However, it can be replaced by the two CIs $\mathsf{HighWindSpeed} \sqsubseteq \mathsf{Storm}$ and $\mathsf{HighWaves} \sqsubseteq \mathsf{Storm}$, while semantic equivalence is retained.

are given in the appendix.

6 Evaluation

We finally evaluate the applicability of the three algorithms for TCQ answering.

To this end, we implemented them prototypically in Java. The correctness was tested using JUnit[5]. The implementations of the IA and the VA overlap in large parts. In particular, they use common representation formats (e.g., for the TCQs and the fact bases) and apply the same (proprietary) implementation of the well-known PerfectRef algorithm [10] to answer CQs over the data (i.e., to evaluate answer terms for these CQs). However, the implementation of the VA (cf. Section 4) requires additional preprocessing for the creation of the vectors; further, the considered window is consumed (and thus to be represented) as a whole, which strongly contrasts the iterative processing of the IA. The RA has been implemented in the QuAnTOn library, which translates TCQs into SQL queries. QuAnTOn accepts TCQs in a format that extends the W3C standard SPARQL[6].

Note that it is not the goal of this paper to evaluate and compare the algorithms exhaustively; instead, we focus on the evaluation of the main characteristics by regarding the time needed for preprocessing and the query answering performance.

6.1 Overview

Our experiments aim to show the following:

- The iterative processing of the IA and our constant domain assumption suit a streaming scenario with the consideration of a fix set of sensors; specifically, the IA then shows a rather constant performance.
- The Vector Algorithm is applicable for window-based TCQ answering.
- The simple rewriting approach of the RA is not applicable for all TCQs, in practice.

Note that an investigation of the effectiveness of the VA, meaning the dependence of the quality of the answers in dependence of different window range and slide values—which is strongly application-specific—, is out of the scope of this paper.

We focus on the querying of weather data as outlined above. We chose this scenario, sketched in [20], for the evaluation of stream reasoning systems, to facilitate a comparison with such systems, which is planned for future work.

[5] `http://junit.org`
[6] `http://www.w3.org/TR/sparql11-query/`

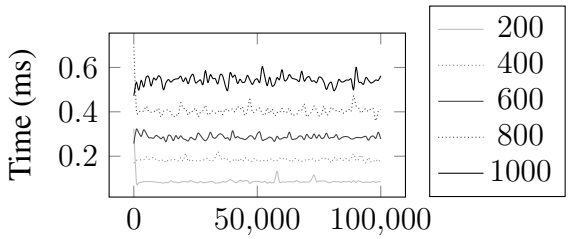

Figure 1: The time the IA takes for answering a TCQ in dependence of the number $n + 1$ of fact bases, for different numbers a of assertions per fact base.

Specifically, we regard sequences of fact bases each containing a specific number a of assertions representing sensor measurements.[7] We experimented with different numbers a of assertions per fact base and correspondingly used one tuple per assertion in the table/database we applied for the evaluation of the RA. We further considered sequences of different length and windows of different range, for the iterative and the window-based processing, respectively. To learn about the performance of the implementations regarding different kinds of TCQs, we only used rather simple CQs within the latter. The ontology we applied extends the sensor-observation ontology[8].

The tests were run on a usual office PC using an 2,2 GHz Intel Core i7 machine with 4 GB RAM, a Java 1.7.0_51 environment, and running OS X 10.8.5. We further applied a MySQL (v5.2.38) database, for answering the queries of the RA.

6.2 Results

Since the iterative processing of the IA contrasts the window-based processing of VA, the algorithms are not directly comparable. We therefore discern these two settings.

Preprocessing For the preprocessing per TCQ, we measured an average time of less than 1 ms for both the IA and the VA, and the times never exceeded 2 ms. For the RA, the preprocessing time per TCQ lay between 100 and 200 ms, in all experiments.

Regarding the Whole History Figure 2 shows that the IA performs rather constant, independent of the considered number of assertions in the fact base. For example, it can be seen that it takes about half a millisecond to (continuously) answer a TCQ over a TKB containing up to $100 * 10^6$ assertions. Thereby, an increase of the number of assertions by 200 causes the runtime to increase by about 0.1 ms. In contrast, the results we obtained for the RA were not as satisfactory. While most TCQs lead to a linear

[7] Note that a can be compared to the tuple-rate (i.e., the rate in which new data is coming into the system—measured in (data) tuples per second, for example), which is usually considered in evaluations of stream reasoning systems.

[8] `http://wiki.knoesis.org/index.php/LinkedSensorData`

increase in runtime (i.e., in dependence of the number of fact bases), which would be acceptable, the operators $\vee$, U, and S caused a quadratic increase. Furthermore, the accessing of the database itself (e.g., the fetching of the results) lead to runtimes in the range of seconds, already for very short sequences of fact bases. We next give examples for this.

Window-based Query Answering Figure 4 shows the runtimes we measured for the window-based query answering. Note that the parameters we considered are also regarded in related work [12]. It can be seen that the VA achieves a linear performance. For a window containing 30 fact bases, for example, the query answering can be done within 4 ms—even for the relatively high number of 1000 assertions per fact base. The figure also illustrates the quadratic increase in runtime we obtained for the RA, as well as its runtimes.

Please note that our investigation is far from being exhaustive. To obtain more general results, further studies should target other application scenarios, consider more complex queries, and directly compare our implementations to the one presented in [12].

7 Conclusions

In this paper, we investigated three algorithms for answering TCQs w.r.t. *DL-Lite* ontologies. In particular, we implemented the Iterative Algorithm, which takes into account all the data given, and a window-based adaptation of the IA, the Vector Algorithm. We further considered a simple Rewriting Algorithm translating TCQs into SQL queries. Our evaluation showed convincing runtimes for the IA and the VA[9] both answering the TCQs within milliseconds. In particular, the constant performance of the IA w.r.t. a growing number of data over time practically confirmed the bounded history

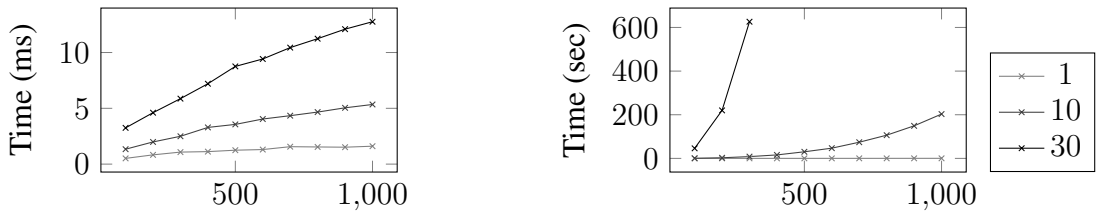

Figure 2: The times the VA (left) and RA (right) take for answering a TCQ in dependence of the number a of assertions per fact base, for three different window ranges (i.e., number of fact bases per window).

[9] According to [12], response times of about 80 ms are still acceptable in a window-based streaming scenario with the parameters we considered—note that our experiments for the VA yielded times which are an order of magnitude less than that.

encoding of the algorithm. The results we obtained for the Rewriting Algorithm show however that the simple approach of rewriting cannot be applied to answer all kinds of TCQs in acceptable times. For a practical application, the RA thus still needs to be optimized; the costs of the disk access also restrict possible applications. Nevertheless, in several applications large amounts of temporal data are stored on disk. It thus would be worth investigating optimizations of the RA.

We further plan a comparison of our implementations with Morph-streams. It will be especially interesting to see if the VA outperforms Morph-streams, which is also window-based but, similar to the RA, delegates the query answering to other systems (i.e., after a translation of the queries).

Future plans also include an extension of our implementations of the IA and the VA such that they support temporal queries based on other rewritable (atemporal) query languages, as it is proposed by [8].[10] For example, temporal queries over TKBs in the DL $\mathcal{EL}$ could be answered by employing a corresponding system for the atemporal query answering.

Acknowlegements.

We thank Stefan Borgwardt for the many helpful discussions on the topics of this paper.

References

[1] Anicic, D., Rudolph, S., Fodor, P., Stojanovic, N.: Stream reasoning and complex event processing in etalis. Semant. web 3(4), 397–407 (Oct 2012)

[2] Artale, A., Kontchakov, R., Lutz, C., Wolter, F., Zakharyaschev, M.: Temporalising tractable description logics. In: Goranko, V., Wang, X.S. (eds.) Proceedings of the 14th International Symposium on Temporal Representation and Reasoning (TIME 2007), pp. 11–22. IEEE Press (2007)

[3] Artale, A., Kontchakov, R., Ryzhikov, V., Zakharyaschev, M.: A cookbook for temporal conceptual data modelling with description logics. ACM Transactions on Computational Logic 15(3), 25 (2014)

[4] Baader, F., Borgwardt, S., Lippmann, M.: Temporalizing ontology-based data access. In: Bonacina, M.P. (ed.) Proc. of the 24th Int. Conf. on Automated De-

[10] Specifically, [8] propose algorithms for answering temporal queries (w.r.t. TKBs) that generalize TCQs in that they combine queries of a generic atemporal query language $\mathcal{Q}$ via LTL-operators (i.e., we restrict $\mathcal{Q}$ to CQs). Thereby, $\mathcal{Q}$ is assumed to be a *rewritable* query language, which basically means that the certain answers to every query in $\mathcal{Q}$ w.r.t. a knowledge base $\mathcal{K}$ can be obtained by answering an appropriate adaptation of the query in a canonical model of $\mathcal{K}$.

duction (CADE'13). Lecture Notes in Computer Science, vol. 7898, pp. 330–344. Springer-Verlag (2013)

[5] Baader, F., Borgwardt, S., Lippmann, M.: Temporal query entailment in the description logic $\mathcal{SHQ}$. Journal of Web Semantics (2015)

[6] Baader, F., Ghilardi, S., Lutz, C.: LTL over description logic axioms. ACM Transactions on Computational Logic 13(3), 21:1–21:32 (2012)

[7] Balduini, M., Calbimonte, J.P., Corcho, O., Dell'Aglio, D., Della Valle, E.: Stream reasoning for linked data, http://streamreasoning.org/events/sr4ld2014

[8] Borgwardt, S., Lippmann, M., Thost, V.: Temporalizing rewritable query languages over knowledge bases. Journal of Web Semantics (2015), in press.

[9] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R.: Ontologies and databases: The dl-lite approach. In: Tessaris, S., Franconi, E., Eiter, T., Gutierrez, C., Handschuh, S., Rousset, M.C., Schmidt, R. (eds.) Reasoning Web. Semantic Technologies for Information Systems, Lecture Notes in Computer Science, vol. 5689, pp. 255–356. Springer Berlin Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-03754-2_7

[10] Calvanese, D., De Giacomo, G., Lemho, D., Lenzerini, M., Rosati, R.: *DL-Lite*: Tractable description logics for ontologies. In: Proceedings of the 20th National Conference on Artificial Intelligence - Volume 2. pp. 602–607. AAAI'05, AAAI Press (2005)

[11] Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: Proc. of the 9th Annual ACM Symp. on Theory of Computing (STOC'77). pp. 77–90. ACM (1977)

[12] Corcho, O., Calbimonte, J.P., Jeung, H., Aberer, K.: Enabling query technologies for the semantic sensor web. Int. J. Semant. Web Inf. Syst. 8(1), 43–63 (Jan 2012)

[13] Cugola, G., Margara, A.: Tesla: A formally defined event specification language. In: Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems. pp. 50–61. DEBS '10, ACM, New York, NY, USA (2010)

[14] Gonçalves, R.S., Bail, S., Jimenez-ruiz, E., Parsia, B., Glimm, B., Kazakov, Y.: OWL Reasoner Evaluation (ORE) workshop 2013 results: Short report

[15] Gutiérrez-Basulto, V., Jung, J.C., Schneider, T.: Lightweight description logics and branching time: A troublesome marriage. In: Proc. of the 14th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'14). AAAI Press (2014)

[16] Holste, J.: Ontology-Based Temporal Reasoning on Streams. Master's thesis, Hamburg University of Technology (October 2014), http://www.sts.tuhh.de/pw-and-m-theses/2014/holste14a.pdf

[17] Klarman, S., Meyer, T.: Querying temporal databases via OWL 2 QL. In: Kontchakov, R., Mugnier, M.L. (eds.) Web Reasoning and Rule Systems, Lecture Notes in Computer Science, vol. 8741, pp. 92–107. Springer International Publishing (2014)

[18] Özçep, O., Möller, R., Neuenstadt, C.: A stream-temporal query language for ontology based data access. In: Lutz, C., Thielscher, M. (eds.) KI 2014: Advances in Artificial Intelligence, Lecture Notes in Computer Science, vol. 8736, pp. 183–194. Springer International Publishing (2014)

[19] Rodriguez-Muro, M., Calvanese, D.: High performance query answering over *DL-Lite* ontologies. In: Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012 (2012)

[20] Zhang, Y., Duc, P.M., Corcho, O., Calbimonte, J.P.: SRBench: a streaming RDF/SPARQL benchmark. In: Proceedings of the 11th International Conference on The Semantic Web - Volume Part I. pp. 641–657. ISWC'12, Springer-Verlag, Berlin, Heidelberg (2012)

Appendix

In what follows, we provide details about the Rewriting Algorithm, which translates a TCQ ϕ into an SQL query $\mathsf{sql}(\phi)$. For simplicity, we make the following assumptions:

- The database contains a table datatable with a column timestamp, and a tuple in this table, for each time point associated with a fact base. Note that not all the data needs to be stored in this table, but it serves as an overview of the time points to be considered.
- The names of the distinguished variables of the CQs which we use equal the corresponding columns of the database.
- We have the following functions:
  - nextTimeStamp (prevTimeStamp): for a given time point, it returns the next (previous) time point w.r.t. all the considered time points.
  - top: given a tuple of variables, it returns all tuples of the same arity, which can be formed using the individual names occurring in the database.
  - commonVariables: given two CQs, it returns the variables that are distinguished variables in both CQs.
  - For a CQ ψ, the function $\mathsf{sql}(\psi)$ returns the SQL rewriting of ψ as it is proposed in [10]. We specifically assume $\mathsf{sql}(\psi)$ to refer to a column timestamp identifying the fact base over which ψ is evaluated.

- $A(x_1, \ldots, x_a) \wedge B(y_1, \ldots, y_b)$

```
SELECT
VIEW_A.x1,...,VIEW_A.xa,
// skip variables that appear before
VIEW_B.y1,...,VIEW_B.yb
FROM sql(A) AS VIEW_A JOIN sql(B) AS VIEW_B
USING (commonVariables(A,B));
```

- $A(x_1, \ldots, x_a) \vee B(y_1, \ldots, y_b)$

```
SELECT
VIEW_A.x1,...,VIEW_A.xa,
// skip variables that appear before
DVIEW_A.y1,...,DVIEW_A.yb
FROM sql(A) AS VIEW_A JOIN top(y1,...,yb) AS DVIEW_A
USING (commonVariables(A,B))
UNION
SELECT
```

```
VIEW_B.x_1,...,VIEW_B.x_a,
DVIEW_B.y_1,...,DVIEW_B.y_b,
FROM sql(B) AS VIEW_B JOIN top(x_1,...,x_a) AS DVIEW_B
USING (commonVariables(A,B));
```

- $\bigcirc A(x_1, \ldots, x_a)$

```
SELECT
//filter out timestamp variable
VIEW_A.x1,...,VIEW_A.xa,
// to refer to the time point where ○A(x1,...,xa) holds
TVIEW_A.timestamp
FROM sql(A) AS VIEW_A,
 (SELECT DISTINCT timestamp FROM datatable) AS TVIEW_A
WHERE
  TVIEW_A.timestamp = prevTimeStamp(VIEW_A.timestamp);
```

For the $\bigcirc^-$-operator, the SQL is correspondingly.

- $A(x_1, \ldots, x_a) \mathsf{U} B(y_1, \ldots, y_b)$

```
SELECT
VIEW_B.y1,...,VIEW_B.yb,
DVIEW_B.x1,...,DVIEW_B.xa,
FROM sql(B) AS VIEW_B JOIN top(x1,...,xa) AS DVIEW_B
USING (commonVariables(A,B))
UNION
SELECT
VIEW_A.x1,...,VIEW_A.xa,
// skip variables that appear before
// esp. timestamp variable is used from A
VIEW_B.y1,...,VIEW_B.yb
FROM sql(A) AS VIEW_A, sql(B) AS VIEW_B
WHERE VIEW_A.timestamp <= VIEW_B.timestamp
AND NOT EXISTS (
        // regard time between A & B
        SELECT TVIEW_C.timestamp
        FROM (SELECT DISTINCT timestamp FROM datatable)
                AS TVIEW_C
        WHERE TVIEW_C.timestamp >= VIEW_A.timestamp
        AND TVIEW_C.timestamp < VIEW_B.timestamp
        AND TVIEW_C.timestamp NOT IN (
                SELECT VIEW_C.timestamp
                FROM sql(A) AS VIEW_C
                WHERE VIEW_A.x1 = VIEW_C.x1 AND ...
                AND VIEW_A.xa = VIEW_C.xa));
```

Again, for the S-operator, the SQL is correspondingly. To optimize the execution, we consider the operators $\Diamond$ and $\Diamond^-$ separately.

- $\Diamond A(x_1, \ldots, x_a)$

```
SELECT
// filter out timestamp variable
VIEW_A.x1,...,VIEW_A.xa,
// to refer to the time point where ◇A(x1,...,xa) holds
TVIEW_A.timestamp
FROM sql(A) AS VIEW_A,
 (SELECT DISTINCT timestamp FROM datatable) AS TVIEW_A
WHERE TVIEW_A.timestamp <= VIEW_A.timestamp;
```

  Again, for the $\Diamond^-$-operator, the SQL is generated correspondingly (i.e., we use '$\geq$' instead of '$\leq$').

- $\Box A(x_1, \ldots, x_a)$

```
SELECT
// filter out timestamp variable
VIEW_A.x1,...,VIEW_A.xa,
// to refer to the time point where □A(x1,...,xa) holds
TVIEW_A.timestamp
FROM sql(A) AS VIEW_A,
 (SELECT DISTINCT timestamp FROM datatable) AS TVIEW_A
WHERE TVIEW_A.timestamp <= VIEW_A.timestamp
AND NOT EXISTS (
        SELECT TVIEW_B.timestamp
        FROM (SELECT DISTINCT timestamp FROM datatable)
                 AS TVIEW_B
        WHERE TVIEW_A.timestamp <= TVIEW_B.timestamp
        AND TVIEW_B.timestamp NOT IN (
                SELECT VIEW_B.timestamp
                FROM sql(A) AS VIEW_B
                WHERE VIEW_A.x1 = VIEW_B.x1 AND ...
                AND VIEW_A.xa = VIEW_B.xa));
```

  Again, for the $\Box^-$-operator, the SQL is similar.