Temporal Query Answering in the Description Logic *DL-Lite*[*]

Stefan Borgwardt, Marcel Lippmann, and Veronika Thost

Theoretical Computer Science, TU Dresden, Germany
{stefborg,lippmann,thost}@tcs.inf.tu-dresden.de

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. For ontologies formulated in Description Logics of the *DL-Lite* family, OBDA can be realized by rewriting the query into a classical first-order query, e.g. an SQL query, by compiling the information of the ontology into the query. The query is then answered using classical database techniques.
In this paper, we consider a temporal version of OBDA. We propose a temporal query language that combines a linear temporal logic with queries over $DL\text{-}Lite_{core}$-ontologies. This language is well-suited to express temporal properties of dynamical systems and is useful in context-aware applications that need to detect specific situations. Using a first-order rewriting approach, we transform our temporal queries into queries over a temporal database. We then present three approaches to answering the resulting queries, all having different advantages and drawbacks.

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]. In general, we cannot assume, however, that the sensors provide a complete description of the current state of the environment. For example, a sensor for certain information might not be available for a moment or not even exist. 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 it is unknown whether they hold or not.

[*] Partially supported by DFG SFB 912 (HAEC) and GRK 1763 (QuantLA).

In addition, though a complete specification of the environment usually does not exist, often some knowledge about its behavior is available. This knowledge can be used to formulate constraints on the interpretation of the predicates used in the queries, to detect more complex situations. In ontology-based data access (OBDA) [10], domain knowledge is encoded in ontologies using a Description Logic (DL). In this paper, we consider logics of the *DL-Lite* family, which are light-weight DLs with a low complexity for many reasoning problems [10]. This low complexity is due to the fact that reasoning problems in *DL-Lite* can often be reduced to answering a first-order query over a relational database.

In order to recognize situations that evolve over time, we propose to add a temporal logical component to the queries. We use the operators of the temporal logic LTL, which allows to reason about a linear and discrete flow of time [20]. 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 asks for ϕ to be satisfied at some point in the future, and *always* ($\Box\phi$), which 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 collection of servers providing several services. An important task is to migrate services between servers to balance the load. To decide when to migrate, we want to detect certain critical situations. We consider a process to be critical if it has an increasing workload, and at the same time the server it is running on is almost overloaded. Suppose that we want to detect those processes and servers that were in a critical situation at least twice within the past ten time units. This can be expressed by the query $\bigcirc^{-10}(\Diamond(\mathsf{Critical}(x,y) \wedge \bigcirc\Diamond\mathsf{Critical}(x,y)))$, where

$$\mathsf{Critical}(x,y) := \mathsf{Server}(x) \wedge \mathsf{Process}(y) \wedge \mathsf{executes}(x,y) \wedge \mathsf{Running}(y) \wedge \mathsf{IncreasingWorkload}(y) \wedge \mathsf{AlmostOverloaded}(x).$$

In this example, it is essential that future and past operators can be nested arbitrarily. 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. We will even show that in our setting it is indeed always possible to construct an equivalent query using only past operators. 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 query results in a blowup that is at least exponential and no constructions of size less than triply exponential are known [18].

Temporal extensions of *DL-Lite* [11] have been considered in the context of conceptual modeling [2,3,4], where the focus lies on checking concept satisfiability instead of query answering. Investigations of temporalized OBDA, the second major use case of *DL-Lite*, with temporal query answering as the most important reasoning problem [10], have started only quite recently. In [16], a framework is developed that combines conjunctive queries in an arbitrary DL and the temporal logic LTL. The algorithm for query answering in this setting is an LTL-satisfiability test using a sub-procedure to answer (atemporal) CQs.

In [6], a similar query language, a combination of LTL and CQs over the DL $\mathcal{ALC}$, is proposed. In contrast to [16], its temporal component is allowed to influence the DL queries via the notion of rigid names, which are names whose interpretation does not change over time. The complexity increases depending on whether only rigid concept names or also rigid role names are allowed. Additionally, the latter paper also studies the so-called data complexity, where the complexity is measured only w.r.t. the size of the sensor data, i.e. the observations, but not w.r.t. the size of the query or the ontology. Another recent paper [5] examines temporal query answering in an extension of *DL-Lite* in which linear temporal operators are allowed to occur inside DL concepts, and proves first-order rewritability for query answering in this logic.

In this paper, we follow an approach suggested in [16] to combine the first-order rewriting techniques for atemporal query answering in logics of the *DL-Lite* family with a temporal component. The main idea is to use optimized database techniques to answer the actual queries. However, the existing techniques for answering temporal queries over temporal databases do not perfectly suit our purposes. In [14], the authors describe a temporal extension of the SQL query language that can answer temporal queries over a complete temporal database. However, in our setting the database containing all previous observations may grow huge very fast, but not all past observations are relevant for a particular query. In [13], an approach is described that reduces the amount of space needed; but the query language considered there allows only for past operators. In addition to describing how these approaches can be applied to our problem, we propose a new algorithm that extends the one from [13] and can also deal with future operators. All three approaches have different advantages and drawbacks.

Additionally, we show how the new algorithm can be extended to deal with rigid concept names for a specific subclass of queries. Unfortunately, there seems to be no simple way to adapt the algorithm to deal with rigid role names.

This paper is an extension of the recently appeared workshop paper [8]. The formal proofs of our results can be found in the technical report [9].

2 Preliminaries

We first describe the DL component, and then the temporal component of our query language. The *DL-Lite* family consists of various DLs that are tailored towards conceptual modeling and allow to realize query answering using classical database techniques. We only consider *DL-Lite*$_{core}$ as a prototypical example.

Definition 1. *Let* $\mathsf{N_C}$, $\mathsf{N_R}$, *and* $\mathsf{N_I}$ *be non-empty, pairwise disjoint sets of* concept, role, *and* individual names, *respectively. A* role expression *is either a role name* $P_1 \in \mathsf{N_R}$ *or an* inverse role P_2^- *with* $P_2 \in \mathsf{N_R}$. *A* basic concept *is of the form* A *or* $\exists R$, *where* $A \in \mathsf{N_C}$ *and* R *is a role expression. A* general concept *is of the form* B *or* $\neg B$, *where* B *is a basic concept.*

A concept inclusion *is of the form* $B \sqsubseteq C$, *where* B *is a basic concept and* C *is a general concept. An* assertion *is of the form* $A(a)$ *or* $P(a, b)$, *where* $A \in \mathsf{N_C}$,

$P \in \mathsf{N_R}$, *and* $a, b \in \mathsf{N_I}$. *A* TBox *(or* ontology*) is a finite set of concept inclusions, and an* ABox *is finite set of assertions.*

The semantics of *DL-Lite*$_{core}$ is defined through the notion of an interpretation.

Definition 2. *An* interpretation *is a pair* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$*, where* $\Delta^{\mathcal{I}}$ *is a non-empty set (called* domain*) and* $\cdot^{\mathcal{I}}$ *is a function that assigns to every* $A \in \mathsf{N_C}$ *a set* $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$*, to every* $P \in \mathsf{N_R}$ *a binary relation* $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$*, and to every* $a \in \mathsf{N_I}$ *an element* $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$*. This function is extended to role expressions, basic concepts, and general concepts as follows:* $(P^-)^{\mathcal{I}} := \{(e, d) \mid (d, e) \in P^{\mathcal{I}}\}$*,* $(\exists R)^{\mathcal{I}} := \{d \mid$ *there is an* $e \in \Delta^{\mathcal{I}}$ *such that* $(d, e) \in R^{\mathcal{I}}\}$*, and* $(\neg C)^{\mathcal{I}} := \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$*.*

$\mathcal{I}$ *is a* model *of* $B \sqsubseteq C$ *if* $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$*, of* $A(a)$ *if* $a^{\mathcal{I}} \in A^{\mathcal{I}}$*, and of* $P(a, b)$ *if* $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in P^{\mathcal{I}}$*. We write* $\mathcal{I} \models \mathcal{T}$ *if* $\mathcal{I}$ *is a model of all concept inclusions in the TBox* $\mathcal{T}$*, and* $\mathcal{I} \models \mathcal{A}$ *if* $\mathcal{I}$ *is a model of all assertions in the ABox* $\mathcal{A}$*. An ABox* $\mathcal{A}$ *is* consistent *(w.r.t. a TBox* $\mathcal{T}$*) if there is an* $\mathcal{I}$ *with* $\mathcal{I} \models \mathcal{A}$ *and* $\mathcal{I} \models \mathcal{T}$*.*

We assume that all interpretations $\mathcal{I}$ satisfy the *unique name assumption* (UNA), i.e. for all $a, b \in \mathsf{N_I}$ with $a \neq b$, we have $a^{\mathcal{I}} \neq b^{\mathcal{I}}$.

We now introduce the notion of temporal knowledge bases. Intuitively, they contain sensor data (ABoxes) for all previous time points, and a global TBox.

Definition 3. *A* temporal knowledge base *(TKB)* $\mathcal{K} = \langle (\mathcal{A}_i)_{0 \leq i \leq n}, \mathcal{T} \rangle$ *consists of a finite sequence of ABoxes* $\mathcal{A}_i$ *and a TBox* $\mathcal{T}$*, where the ABoxes* $\mathcal{A}_i$ *can only contain concept names that also occur in* $\mathcal{T}$*. Let* $\mathfrak{I} = (\mathcal{I}_i)_{0 \leq i \leq n}$ *be a sequence of interpretations* $\mathcal{I}_i = (\Delta, \cdot^{\mathcal{I}_i})$ *over a fixed non-empty domain* Δ*. Then* $\mathfrak{I}$ *is a* model *of* $\mathcal{K}$ *(written* $\mathfrak{I} \models \mathcal{K}$*) if* $\mathcal{I}_i \models \mathcal{A}_i$ *and* $\mathcal{I}_i \models \mathcal{T}$ *for all* i*,* $0 \leq i \leq n$*.*

Similar to what was done in [6,16], our temporal query language is based on conjunctive queries [1,12]. The main difference is that we do not allow for negation, as in *DL-Lite* arbitrary negation is disallowed. In contrast to [5], we also do not allow temporal operators inside concepts. These restrictions allow us to apply first-order rewritability of (atemporal) conjunctive queries in a black-box fashion to obtain a similar result for our temporal query language (see Section 3).

Definition 4. *Let* $\mathsf{N_V}$ *be a set of variables. A* conjunctive query *(CQ) is of the form* $\phi = \exists y_1, \ldots, y_m.\psi$*, where* $y_1, \ldots, y_m \in \mathsf{N_V}$ *and* ψ *is a (possibly empty) finite conjunction of* atoms *of the form* $A(z)$ *for* $A \in \mathsf{N_C}$ *and* $z \in \mathsf{N_V} \cup \mathsf{N_I}$ *(*concept atom*); or* $r(z_1, z_2)$ *for* $r \in \mathsf{N_R}$ *and* $z_1, z_2 \in \mathsf{N_V} \cup \mathsf{N_I}$ *(*role atom*). The empty conjunction is denoted by* true*.*

Temporal conjunctive queries *(TCQs) are built from CQs as follows: each CQ is a TCQ, and if* ϕ_1 *and* ϕ_2 *are TCQs, then so are* $\phi_1 \wedge \phi_2$ *(conjunction),* $\phi_1 \vee \phi_2$ *(disjunction),* $\bigcirc \phi_1$ *(strong next),* $\bullet \phi_1$ *(weak next),* $\bigcirc^- \phi_1$ *(strong previous),* $\bullet^- \phi_1$ *(weak previous),* $\phi_1 \,\mathsf{U}\, \phi_2$ *(until), and* $\phi_1 \,\mathsf{S}\, \phi_2$ *(since).*

The symbols $\bigcirc^-$, $\bullet^-$, and S are called *past operators*, the symbols $\bigcirc$, $\bullet$, and U are *future operators*. All results also hold in the presence of the additional temporal operators $\Box$ (always), $\Box^-$ (always in the past), $\Diamond$ (eventually), and $\Diamond^-$ (some time in the past) [9], but we omit them here for space reasons.

We denote the set of individuals occurring in a TCQ ϕ by $\mathsf{Ind}(\phi)$, the set of variables occurring in ϕ by $\mathsf{Var}(\phi)$, the set of free variables in ϕ by $\mathsf{FVar}(\phi)$, and the set of atoms occurring in ϕ by $\mathsf{At}(\phi)$. A TCQ ϕ is called *Boolean* if $\mathsf{FVar}(\phi) = \emptyset$. We further denote by $\mathsf{Sub}(\phi)$ the set of all TCQs occurring as subqueries in ϕ (including ϕ itself). A *union of conjunctive queries* (UCQ) is a disjunction of CQs. For our purposes, it is sufficient to define the semantics for Boolean CQs and TCQs. As usual, it is given using the notion of a homomorphism [12].

Definition 5. *Let $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ be an interpretation and ψ be a Boolean CQ. A mapping $\pi\colon \mathsf{Var}(\psi) \cup \mathsf{N_I} \to \Delta$ is a* homomorphism *of ψ into $\mathcal{I}$ if $\pi(a) = a^{\mathcal{I}}$ for all $a \in \mathsf{N_I}$, $\pi(z) \in A^{\mathcal{I}}$ for all concept atoms $A(z)$ in ψ, and $(\pi(z_1), \pi(z_2)) \in r^{\mathcal{I}}$ for all role atoms $r(z_1, z_2)$ in ψ. We say that $\mathcal{I}$ is a* model *of ψ (written $\mathcal{I} \models \psi$) if there is such a homomorphism.*

Let now ϕ be a Boolean TCQ. For a sequence of interpretations $\mathfrak{I} = (\mathcal{I}_i)_{0 \le i \le n}$ and i with $0 \le i \le n$, we define $\mathfrak{I}, i \models \phi$ by induction on the structure of ϕ:

$\mathfrak{I}, i \models \exists y_1, \dots, y_m.\psi$	*iff* $\mathcal{I}_i \models \exists y_1, \dots, y_m.\psi$
$\mathfrak{I}, i \models \phi_1 \wedge \phi_2$	*iff* $\mathfrak{I}, i \models \phi_1$ *and* $\mathfrak{I}, i \models \phi_2$
$\mathfrak{I}, i \models \phi_1 \vee \phi_2$	*iff* $\mathfrak{I}, i \models \phi_1$ *or* $\mathfrak{I}, i \models \phi_2$
$\mathfrak{I}, i \models \bigcirc \phi_1$	*iff* $i < n$ *and* $\mathfrak{I}, i+1 \models \phi_1$
$\mathfrak{I}, i \models \bullet \phi_1$	*iff* $i < n$ *implies* $\mathfrak{I}, i+1 \models \phi_1$
$\mathfrak{I}, i \models \bigcirc^{-} \phi_1$	*iff* $i > 0$ *and* $\mathfrak{I}, i-1 \models \phi_1$
$\mathfrak{I}, i \models \bullet^{-} \phi_1$	*iff* $i > 0$ *implies* $\mathfrak{I}, i-1 \models \phi_1$
$\mathfrak{I}, i \models \phi_1 \,\mathsf{U}\, \phi_2$	*iff there is some* k, $i \le k \le n$ *such that* $\mathfrak{I}, k \models \phi_2$ *and* $\mathfrak{I}, j \models \phi_1$ *for all* j, $i \le j < k$
$\mathfrak{I}, i \models \phi_1 \,\mathsf{S}\, \phi_2$	*iff there is some* k, $0 \le k \le i$ *such that* $\mathfrak{I}, k \models \phi_2$ *and* $\mathfrak{I}, j \models \phi_1$ *for all* j, $k < j \le i$.

Here we assume that there is no time point before 0 or after n, similar to the temporal semantics used for LTL in [23] or for temporal query languages for databases [13,17,21]. As in classical LTL, one can show that $\phi_1 \,\mathsf{S}\, \phi_2$ is equivalent to $\phi_2 \vee (\phi_1 \wedge \bigcirc^{-}(\phi_1 \,\mathsf{S}\, \phi_2))$, and a similar equivalence holds for U.

We are now ready to introduce the central reasoning problem of this paper, namely to find certain answers to TCQs.

Definition 6. *Let ϕ be a TCQ, $\mathfrak{I} = (\mathcal{I}_i)_{0 \le i \le n}$ a sequence of interpretations, and $i \ge 0$. The mapping $\mathfrak{a}\colon \mathsf{FVar}(\phi) \to \mathsf{N_I}$ is an* answer *to ϕ w.r.t. $\mathfrak{I}$ at time point i if $\mathfrak{I}, i \models \mathfrak{a}(\phi)$, where $\mathfrak{a}(\phi)$ denotes the Boolean TCQ that is obtained from ϕ by replacing the free variables according to $\mathfrak{a}$. Let further $\mathcal{K} = \langle (\mathcal{A}_i)_{0 \le i \le n}, \mathcal{T} \rangle$ be a TKB. A mapping $\mathfrak{a}\colon \mathsf{FVar}(\phi) \to \mathsf{N_I}$ is a* certain answer *to ϕ w.r.t. $\mathcal{K}$ at time point i if for every $\mathfrak{J} \models \mathcal{K}$, we have $\mathfrak{J}, i \models \mathfrak{a}(\phi)$.*

The set of all answers to ϕ w.r.t. $\mathfrak{I}$ at time point i is denoted by $\mathsf{Ans}(\phi, \mathfrak{I}, i)$, and the set of all certain answers to ϕ w.r.t. $\mathcal{K}$ is denoted by $\mathsf{Cert}(\phi, \mathcal{K}, i)$. Recall that our main interest lies in finding answers to queries at the current time point, i.e. computing the sets $\mathsf{Ans}(\phi, \mathfrak{I}) := \mathsf{Ans}(\phi, \mathfrak{I}, n)$ or $\mathsf{Cert}(\phi, \mathcal{K}) := \mathsf{Cert}(\phi, \mathcal{K}, n)$.

We will sometimes use the abbreviation $\mathsf{false} := A(x) \wedge A'(x)$, where A, A' are new concept names for which we assume that the concept inclusion $A \sqsubseteq \neg A'$ is contained in the global TBox $\mathcal{T}$.

3 Answering Temporal Conjunctive Queries

For computing the set of certain answers for a *conjunctive query*, the *rewriting approach* [10] can be employed. It compiles the information contained in the TBox into the query and evaluates the query w.r.t. the ABox (viewed as database) using classical database techniques. A similar approach is possible for TCQs.

Definition 7. *For an ABox $\mathcal{A}$, the interpretation* $\mathsf{DB}(\mathcal{A}) := (\mathsf{N_I}, \cdot^{\mathsf{DB}(\mathcal{A})})$ *is defined as follows:*

- $a^{\mathsf{DB}(\mathcal{A})} := a$ *for all* $a \in \mathsf{N_I}$*;*
- $A^{\mathsf{DB}(\mathcal{A})} := \{a \mid A(a) \in \mathcal{A}\}$ *for all* $A \in \mathsf{N_C}$*; and*
- $P^{\mathsf{DB}(\mathcal{A})} := \{(a, b) \mid P(a, b) \in \mathcal{A}\}$ *for all* $P \in \mathsf{N_R}$.

As shown in [10], this interpretation is the smallest model of $\mathcal{A}$. In order to employ database techniques, we must assume $\mathsf{DB}(\mathcal{A})$, and thus $\mathsf{N_I}$, to be finite.

Proposition 8 ([10]). *Let ψ be a CQ, $\mathcal{A}$ be an ABox, and $\mathcal{T}$ be a TBox. There is a* canonical model $\mathcal{I}_{\mathcal{A},\mathcal{T}}$ *of $\mathcal{A}$ and $\mathcal{T}$ and a UCQ $\psi^{\mathcal{T}}$ such that*

$$\mathsf{Cert}(\psi, \langle \mathcal{A}, \mathcal{T} \rangle) = \mathsf{Ans}(\psi, \mathcal{I}_{\mathcal{A},\mathcal{T}}) = \mathsf{Ans}(\psi^{\mathcal{T}}, \mathsf{DB}(\mathcal{A})).$$

We now use this proposition to show a similar result for TCQs. Let ϕ be a TCQ and $\mathcal{K} = \langle (\mathcal{A}_i)_{0 \leq i \leq n}, \mathcal{T} \rangle$ be a TKB. The TCQ $\phi^{\mathcal{T}}$ is obtained by replacing each CQ ψ occurring in ϕ by $\psi^{\mathcal{T}}$. Note that $\phi^{\mathcal{T}}$ is again a TCQ since $\psi^{\mathcal{T}}$ is always a UCQ. Let furthermore $\mathfrak{I}_{\mathcal{K}} := (\mathcal{I}_{\mathcal{A}_i,\mathcal{T}})_{0 \leq i \leq n}$ and $\mathsf{DB}(\mathcal{K}) := (\mathsf{DB}(\mathcal{A}_i))_{0 \leq i \leq n}$. The following theorem can be shown by a straightforward induction on the structure of ϕ.

Theorem 9. *For every TCQ ϕ, TKB $\mathcal{K} = \langle (\mathcal{A}_i)_{0 \leq i \leq n}, \mathcal{T} \rangle$, and $i \geq 0$, we have* $\mathsf{Cert}(\phi, \mathcal{K}, i) = \mathsf{Ans}(\phi, \mathfrak{I}_{\mathcal{K}}, i) = \mathsf{Ans}(\phi^{\mathcal{T}}, \mathsf{DB}(\mathcal{K}), i)$.

More importantly, for every TCQ ϕ and TKB $\mathcal{K} = \langle (\mathcal{A}_i)_{0 \leq i \leq n}, \mathcal{T} \rangle$, it holds that $\mathsf{Cert}(\phi, \mathcal{K}) = \mathsf{Ans}(\phi^{\mathcal{T}}, \mathsf{DB}(\mathcal{K}))$. It thus remains to show how to compute the set $\mathsf{Ans}(\phi, \mathfrak{I})$ for a TCQ ϕ and a sequence $\mathfrak{I} = (\mathcal{I}_i)_{0 \leq i \leq n}$ of interpretations over a finite domain. A first possibility is to view $\mathfrak{I}$ as a temporal database and rewrite ϕ into an ATSQL query [14]. However, since our goal is to monitor processes that produce new data in very short time intervals, storing all the data for all previous time points is not feasible. Therefore, we describe two different approaches that reduce the amount of space necessary to compute $\mathsf{Ans}(\phi, \mathfrak{I})$. Since we are interested in the answers at the last time point, the idea is to keep only the past information necessary to answer the query ϕ.

In the first approach (Section 4), we rewrite ϕ into a TCQ ϕ' without future operators, employing a construction described in [15]. We then compute $\mathsf{Ans}(\phi', \mathfrak{I})$ using an algorithm described in [13,22] that uses a so-called *bounded history encoding*, which means that the space required by the algorithm is constant w.r.t. the number n of previous time points. Only the current state of the database and some auxiliary relations have to be stored.

In Section 5, we generalize the algorithm from [13] to directly deal with the future operators. The main difference is that we do not consider negation or arbitrary first-order queries. Unfortunately, the space required by this algorithm is in general exponential in n and thus does not constitute a bounded history encoding in the sense of [13,22]. However, it allows us to circumvent the non-elementary blow-up of the formula resulting from the reduction in [15].

4 Eliminating Future Operators

To rewrite a TCQ ϕ into an equivalent TCQ that does not contain future operators, we employ the separation theorem for propositional LTL [15]. We describe here only the general idea, details can be found in the technical report [9].

The separation theorem cannot be applied directly since our temporal semantics differs from that in [15]: the only temporal operators in [15] are strict versions of U and S, and the semantics is defined w.r.t. bounded past and unbounded future. To apply this theorem, we replace the CQs in ϕ by propositional variables, rewrite U and S into their restrict counterparts, and use an additional propositional variable to delimit the time interval from 0 to n.

We can then apply the separation theorem to the resulting LTL-formula $\widehat{\phi}$. We obtain an equivalent LTL-formula $\widehat{\phi}'$ with negation which is a Boolean combination of temporal subformulae that either contain only strict S operators or only strict U operators. In this construction, subformulae of $\widehat{\phi}$ are copied and rearranged, but no additional propositional variables are introduced.

Since we are interested in evaluating ϕ at n, we can replace all variables in $\widehat{\phi}'$ that are in the scope of a strict U by false. The reason for this is that such variables are only evaluated at time points after n, where all variables are false. The resulting formula is then simplified to eliminate all strict U operators, and then translated back into a Boolean TCQ ϕ' by replacing the propositional variables by the corresponding CQs. Note that ϕ' contains no future operators.

We then apply the algorithm described in [13] to iteratively compute the answers to ϕ' at each time point.[1] The main advantage of this approach is that we can compute this set *iteratively* and such that the required memory is independent of the length of the sequence $\mathfrak{I}$. More formally, let $\mathfrak{I} = (\mathcal{I}_i)_{i \geq 0}$ be an *infinite* sequence of interpretations representing the observations over all time points. In our setting, these interpretations are generated from an infinite sequence of ABoxes that represent the observed sensor data using the construction of Section 3. At each time point $i \geq 0$, we only have access to the finite prefix $\mathfrak{I}^{(i)} := (\mathcal{I}_j)_{0 \leq j \leq i}$ of $\mathfrak{I}$ of length $i+1$. Let Δ be the shared domain of the interpretations in $\mathfrak{I}$.

The algorithm from [13] works on ϕ' as follows. On input $\mathcal{I}_0$, it computes a first-order interpretation $\mathcal{I}'_0$ of several auxiliary predicates. Intuitively, for each subformula ψ of ϕ' beginning with a past operator, the algorithm stores the answers $\mathsf{Ans}(\psi, \mathfrak{I}^{(0)}) \subseteq \Delta^{\mathsf{FVar}(\psi)}$ for ψ in a new relation $A_\psi^{\mathcal{I}'_0}$ of arity $|\mathsf{FVar}(\psi)|$.

[1] Before we can use the algorithm presented in [13], we need another rewriting step since in that paper the semantics of S is slightly different (see [9] for details).

The set $\mathsf{Ans}(\phi', \mathfrak{I}^{(0)})$ can then easily be computed from $\mathcal{I}_0$ and $\mathcal{I}'_0$. Afterwards, the algorithm disregards $\mathcal{I}_0$ and keeps only the information computed in $\mathcal{I}'_0$. On input $\mathcal{I}_1$, it then updates $\mathcal{I}'_0$ to a new interpretation $\mathcal{I}'_1$, which allows it to compute $\mathsf{Ans}(\phi', \mathfrak{I}^{(1)})$, and so on.

The memory requirements of this algorithm are bounded polynomially in the size of Δ, in the number of concept and role names, and in the number of past operators occurring in ϕ', and exponentially in the number of free variables occurring below past operators. However, the memory requirements do not depend on the length of the sequence of interpretations seen so far. This is called a *bounded history encoding* in [13].

Overall, the presented approach has, however, several drawbacks. First, the rewritings from ϕ to $\widehat{\phi}$ and from $\widehat{\phi}'$ to ϕ' may duplicate subformulae, which can cause exponential blowups in the size of ϕ. This could be avoided by applying a reduction similar to the one for propositional LTL in [15] directly to ϕ. However, since the reduction in [15] is already non-elementary in the size of the formula, this is not much more efficient. Hence, the presented approach is best suited for answering simple, small queries ϕ over large databases.

5 A New Algorithm

In this section, we present an algorithm that computes the set $\mathsf{Ans}(\phi, \mathfrak{I})$ without the need to eliminate the future operators beforehand, thereby avoiding the non-elementary blowup of the construction described in the previous section. However, the memory requirements of this new algorithm are not independent of the number of previous time points. From now on, let ϕ be a fixed TCQ and $\mathfrak{I} = (\mathcal{I}_i)_{i \geq 0}$ be a fixed infinite sequence of interpretations over the same finite domain Δ. For $i \geq 0$, we denote by $\mathfrak{I}^{(i)} := (\mathcal{I}_j)_{0 \leq j \leq i}$ the finite prefix of $\mathfrak{I}$ of length $i+1$. Our algorithm iteratively computes the sets $\mathsf{Ans}(\phi, \mathfrak{I}^{(i)})$. It uses as data structure so-called *answer formulae*, which represent TCQs in which some parts have already been evaluated. In particular, they do not contain CQs any more, but sets of already computed answers to subqueries. Additionally, they may contain variables (different from those in $\mathsf{N_V}$) that serve as place-holders for subqueries that have to be evaluated at the next time point.

For ease of presentation, we assume in the following that $\mathsf{N_V}$ is finite and that answers are of the form $\mathfrak{a} \colon \mathsf{N_V} \to \Delta$ instead of $\mathfrak{a} \colon \mathsf{FVar}(\phi) \to \Delta$. Thus, when we talk about answers, we mean mappings $\mathfrak{a} \colon \mathsf{N_V} \to \Delta$, and in particular $\mathsf{Ans}(\ldots)$ refers to a set of such mappings, i.e. a subset of $\Delta^{\mathsf{N_V}}$.[2]

Definition 10. *Let* $\mathsf{FSub}(\phi)$ *denote the set of all subqueries of* ϕ *of the form* $\bigcirc\psi_1$, $\bullet\psi_1$, *or* $\psi_1 \,\mathsf{U}\, \psi_2$. *For* $j \geq 0$, *we denote by* Var_j^ϕ *the set of all variables of the form* x_j^ψ *for* $\psi \in \mathsf{FSub}(\phi)$. *The set* AF_ϕ^i *of all* answer formulae for ϕ at $i \geq 0$ *is the smallest set satisfying the following conditions:*

[2] In an implementation, one should restrict the intermediate computations of answers for subqueries ψ to $\mathsf{FVar}(\psi)$. But then one has to be more careful when combining answers to different subqueries.

Table 1. Computing answer formulae for a TCQ

ϕ	$\Phi_0(\phi)$	$\Phi_i^0(\phi)$
CQ ψ_1	$\mathsf{Ans}(\psi_1, \mathfrak{I}^{(0)})$	$\mathsf{Ans}(\psi_1, \mathfrak{I}^{(i)})$
$\psi_1 \wedge \psi_2$	$\Phi_0(\psi_1) \cap \Phi_0(\psi_2)$	$\Phi_i^0(\psi_1) \cap \Phi_i^0(\psi_2)$
$\psi_1 \vee \psi_2$	$\Phi_0(\psi_1) \cup \Phi_0(\psi_2)$	$\Phi_i^0(\psi_1) \cup \Phi_i^0(\psi_2)$
$\bigcirc\psi_1$	$x_0^{\bigcirc\psi_1}$	$x_i^{\bigcirc\psi_1}$
$\bigcirc^-\psi_1$	$\emptyset$	$\Phi_{i-1}(\psi_1)$
$\bullet\psi_1$	$x_0^{\bullet\psi_1}$	$x_i^{\bullet\psi_1}$
$\bullet^-\psi_1$	$\Delta^{\mathsf{N_V}}$	$\Phi_{i-1}(\psi_1)$
$\psi_1 \,\mathsf{U}\, \psi_2$	$\Phi_0(\psi_2) \cup (\Phi_0(\psi_1) \cap x_0^{\psi_1 \,\mathsf{U}\, \psi_2})$	$\Phi_0(\psi_2) \cup (\Phi_i^0(\psi_1) \cap x_i^{\psi_1 \,\mathsf{U}\, \psi_2})$
$\psi_1 \,\mathsf{S}\, \psi_2$	$\Phi_0(\psi_2)$	$\Phi_i^0(\psi_2) \cup (\Phi_i^0(\psi_1) \cap \Phi_{i-1}(\psi_1 \,\mathsf{S}\, \psi_2))$

- *Every set $A \subseteq \Delta^{\mathsf{N_V}}$ is an answer formula for ϕ at i.*
- *Every $x_j^\psi \in \mathsf{Var}_j^\phi$ with $j \leq i$ is an answer formula for ϕ at i.*
- *If α_1 and α_2 are answer formulae for ϕ at i, then so are $\alpha_1 \cap \alpha_2$ and $\alpha_1 \cup \alpha_2$.*

In order to evaluate these answer formulae, we introduce the notion of correctness. Intuitively, an answer formula α for ϕ at i is correct for i if we obtain the set $\mathsf{Ans}(\phi, \mathfrak{I}^{(i)})$ by replacing the variables x_j^ψ in α by appropriate sets of answers and evaluating $\cap$ and $\cup$ as set intersection and union, respectively.

Definition 11. *We define the function $\mathsf{eval}^n \colon \mathsf{AF}_\phi^n \to 2^{\Delta^{\mathsf{N_V}}}$, $n \geq 0$, as follows:*

- $\mathsf{eval}^n(A) := A$ *if* $A \subseteq \Delta^{\mathsf{N_V}}$*;*
- $\mathsf{eval}^n(x_j^\psi) := \begin{cases} \mathsf{Ans}(\psi_1, \mathfrak{I}^{(n)}, j+1) & \text{if } j < n \text{ and } \psi = \bigcirc\psi_1 \text{ or } \psi = \bullet\psi_1; \\ \mathsf{Ans}(\psi, \mathfrak{I}^{(n)}, j+1) & \text{if } j < n \text{ and } \psi = \psi_1 \,\mathsf{U}\, \psi_2; \\ \emptyset & \text{if } j = n \text{ and } \psi = \bigcirc\psi_1 \text{ or } \psi = \psi_1 \,\mathsf{U}\, \psi_2; \\ \Delta^{\mathsf{N_V}} & \text{if } j = n \text{ and } \psi = \bullet\psi_1; \end{cases}$
- $\mathsf{eval}^n(\alpha_1 \cap \alpha_2) := \mathsf{eval}^n(\alpha_1) \cap \mathsf{eval}^n(\alpha_2)$*; and*
- $\mathsf{eval}^n(\alpha_1 \cup \alpha_2) := \mathsf{eval}^n(\alpha_1) \cup \mathsf{eval}^n(\alpha_2)$.

We say that a mapping $\Phi \colon \mathsf{Sub}(\phi) \to \mathsf{AF}_\phi^i$ is correct for $i \geq 0$ *if for all $n \geq i$ and for all $\psi \in \mathsf{Sub}(\phi)$, we have $\mathsf{eval}^n(\Phi(\psi)) = \mathsf{Ans}(\psi, \mathfrak{I}^{(n)}, i)$.*

In particular, if $\Phi \colon \mathsf{Sub}(\phi) \to \mathsf{AF}_\phi^i$ is correct for i, then $\mathsf{eval}^i(\Phi(\phi)) = \mathsf{Ans}(\phi, \mathfrak{I}^{(i)})$, which is the set we want to compute. Note that $x_j^{\psi_1 \,\mathsf{U}\, \psi_2}$ is actually a placeholder for $\bigcirc(\psi_1 \,\mathsf{U}\, \psi_2)$ since we evaluate the U operator according to the recursive equivalence $\psi_1 \,\mathsf{U}\, \psi_2 \equiv \psi_2 \vee (\psi_1 \wedge \bigcirc(\psi_1 \,\mathsf{U}\, \psi_2))$ (cf. Table 1).

The algorithm works as follows. It first computes a mapping Φ_0 that is correct for 0, which is used to compute the next mapping Φ_1 when the interpretation $\mathcal{I}_1$ becomes available. This mapping is correct for 1 and can be used to compute the next mapping Φ_2, and so on. In each step, to compute Φ_{i+1}, we only need Φ_i and the interpretation $\mathcal{I}_{i+1}$. We recursively define the mapping $\Phi_0 \colon \mathsf{Sub}(\phi) \to \mathsf{AF}_\phi^0$ as shown in the second column of Table 1. Here, CQs are answered, e.g. by evaluating them as first-order queries over the database $\mathcal{I}_0$ [1].

Table 2. An example computation

ϕ	ψ_1	ψ_2	$\psi_1 \mathsf{S} \psi_2$
$\Phi_0(\phi)$	$x_0^{\psi_1}$	$B_0 \cup (A_0 \cap x_0^{\psi_2})$	$B_0 \cup (A_0 \cap x_0^{\psi_2})$
$\mathsf{Ans}(\phi, \mathfrak{I}^{(0)})$	$\Delta^{\mathsf{N_V}}$	B_0	B_0
$\Phi_1^0(\phi)$	$x_1^{\psi_1}$	$B_1 \cup (A_1 \cap x_1^{\psi_2})$	$\Phi_1^0(\psi_2) \cup (x_1^{\psi_1} \cap (B_0 \cup (A_0 \cap x_0^{\psi_2})))$
$\Phi_1(\phi)$	$x_1^{\psi_1}$	$B_1 \cup (A_1 \cap x_1^{\psi_2})$	$\Phi_1(\psi_2) \cup (x_1^{\psi_1} \cap (B_0 \cup (A_0 \cap \Phi_1(\psi_2))))$
			$\equiv ((x_1^{\psi_1} \cap B_0) \cup B_1) \cup (A_1 \cap x_1^{\psi_2})$
$\mathsf{Ans}(\phi, \mathfrak{I}^{(1)})$	$\Delta^{\mathsf{N_V}}$	B_1	$B_0 \cup B_1$
$\Phi_2(\phi)$	$x_2^{\psi_1}$	$B_2 \cup (A_2 \cap x_2^{\psi_2})$	$\Phi_2(\psi_2) \cup (x_2^{\psi_1} \cap (((x_1^{\psi_1} \cap B_0) \cup B_1) \cup (A_1 \cap x_1^{\psi_2})))$
			$\equiv ((x_2^{\psi_1} \cap ((C_1 \cap B_0) \cup B_1)) \cup B_2) \cup (A_1 \cap x_2^{\psi_2})$
$\mathsf{Ans}(\phi, \mathfrak{I}^{(2)})$	$\Delta^{\mathsf{N_V}}$	B_2	$(C_1 \cap B_0) \cup B_1 \cup B_2$

Assume now that $\Phi_{i-1} \colon \mathsf{Sub}(\phi) \to \mathsf{AF}_\phi^{i-1}$ is a function containing only variables with index $i-1$. We proceed as follows to construct a new function that contains only variables with index i. We recursively define the mapping $\Phi_i^0 \colon \mathsf{Sub}(\phi) \to \mathsf{AF}_\phi^i$ similarly to Φ_0 as given in the third column of Table 1.

Example 12. Consider the TCQ $\psi_1 \mathsf{S} \psi_2$ with two subqueries referring to the future, $\psi_1 := \bullet C(x)$ and $\psi_2 := A(x) \mathsf{U} B(x)$, and let $A_0 := \mathsf{Ans}(A(x), \mathfrak{I}^{(0)})$, and similarly for the other CQs and time points. The answer formulae Φ_0 and Φ_1^0 are listed in Table 2.

The difference to the definition of Φ_0 is that the answer formulae for past operators are computed using the answer formulae for the previous time point. This means that Φ_i^0 may still contain variables with index $i-1$. We now remove these old variables by substituting them appropriately. For example, since $x_{i-1}^{\bigcirc\psi}$ is a place-holder for the answers to ψ w.r.t. $\mathfrak{I}^{(n)}$ at i, we can now replace it by $\Phi_i^0(\psi)$. However, this formula may itself contain another old variable, and thus we have to be careful about the order in which we do these substitutions. Since each $\Phi_i^0(\psi)$ can contain only variables that refer to subqueries of ψ, by replacing the variables for "smaller" subqueries first, we ensure that all variables with index $i-1$ are eliminated. The details of this construction can be found in [9]. We obtain a mapping $\Phi_i \colon \mathsf{Sub}(\phi) \to \mathsf{AF}_\phi^i$ that is correct for i.

Lemma 13. *For each $i \geq 0$, the mapping Φ_i is correct for i.*

Example 14. Consider again the query $\psi_1 \mathsf{S} \psi_2$ from Example 12. Since $\Phi_1^0(\psi_1)$ and $\Phi_1^0(\psi_2)$ do not contain variables with index 0, the value of Φ_1 is the same as that of Φ_1^0 for both of these subqueries. We only have to replace $x_0^{\psi_2}$ within $\Phi_1^0(\psi_1 \mathsf{S} \psi_2)$ by $\Phi_1(\psi_2)$ to obtain $\Phi_1(\psi_1 \mathsf{S} \psi_2)$ as listed in Table 2. To obtain the answers at time point 1, we can now replace the remaining variables in according to eval^1, which yields $\mathsf{Ans}(\psi_1 \mathsf{S} \psi_2, \mathfrak{I}^{(1)}) = B_0 \cup B_1$.

Consider now the algorithm, which, on input ϕ and $\mathfrak{I}$, computes the mappings Φ_i as described above, and outputs $\mathsf{eval}^i(\Phi_i(\phi))$ for each $i \geq 0$. The following is a trivial consequence of the correctness of these mappings.

Theorem 15. *Given a TCQ ϕ and an infinite sequence $\mathfrak{I} = (\mathcal{I}_i)_{i\geq 0}$ of interpretations, the algorithm outputs $\mathsf{Ans}(\phi, \mathfrak{I}^{(i)})$ for each $i \geq 0$.*

It is easy to compute the sets $\mathsf{eval}^i(\Phi_i(\phi)) = \mathsf{Ans}(\phi, \mathfrak{I}^{(i)})$ for $i \geq 0$ since each of the variables x_i^ψ in $\Phi_i(\phi)$ simply has to be replaced by either $\emptyset$ or $\Delta^{\mathsf{N_V}}$ (see Definition 11). However, as mentioned earlier, the size of the formula $\Phi_i(\phi)$ may depend exponentially on the length i of the current sequence of interpretations.

Example 16. Consider again the query $\psi_1 \,\mathsf{S}\, \psi_2$ from Example 12. After replacing $x_0^{\psi_2}$ by $\Phi_1(\psi_2)$, the variable $x_1^{\psi_2}$ occurs twice in $\Phi_1(\psi_1 \,\mathsf{S}\, \psi_2)$. In general, $\Phi_i(\psi_1 \,\mathsf{S}\, \psi_2)$ will contain 2^i occurrences of the variable $x_i^{\psi_2}$. However, applying the associativity, commutativity, distributivity, and absorption laws for $\cap$ and $\cup$ does not affect the semantics of answer formulae (given by eval), and hence

$$\begin{aligned}
\Phi_1(\phi) &= \Phi_1(\psi_2) \cup (x_1^{\psi_1} \cap (B_0 \cup (A_0 \cap \Phi_1(\psi_2)))) \\
&\equiv \Phi_1(\psi_2) \cup (x_1^{\psi_1} \cap B_0) \cup (x_1^{\psi_1} \cap A_0 \cap \Phi_1(\psi_2)) \\
&\equiv \Phi_1(\psi_2) \cup (x_1^{\psi_1} \cap B_0) \\
&= (B_1 \cup (A_1 \cap x_1^{\psi_2})) \cup (x_1^{\psi_1} \cap B_0) \\
&\equiv ((x_1^{\psi_1} \cap B_0) \cup B_1) \cup (A_1 \cap x_1^{\psi_2})
\end{aligned}$$

The resulting formula contains $x_1^{\psi_2}$ only once. In general, the formula $\Phi_i(\phi)$ is equivalent to $((x_i^{\psi_1} \cap D_i) \cup B_i) \cup (A_i \cap x_i^{\psi_2})$, where $D_0 := \emptyset$ and for $i > 0$, we set $D_{i+1} := (C_{i+1} \cap D_i) \cup B_i$. Thus, the algorithm only has to store the sets $A_i, B_i, D_i \subseteq \Delta^{\mathsf{N_V}}$ at each time point, i.e. we achieve a bounded history encoding as in [13].

If the formula ϕ contains no future operators, then the answer formulae contain no variables and can always be fully evaluated to a subset of $\Delta^{\mathsf{N_V}}$. In this special case, our algorithm can be seen as a variant of the one from [13] for less expressive queries. Example 16 demonstrates that it is important that the computed answer formulae are simplified at each step, while preserving their semantics under eval. However, this does not guarantee a bounded history encoding as in [13].

6 Rigid Names

We now extend our temporal query language by designating certain concept names as being *rigid*, which means that their interpretation is not allowed to change over time. This especially makes sense regarding our application. For example, if the concept name Server describes the set of all severs, then it should be rigid since an application scenario with a server that stops being a server at some point in time would make no sense. The notion of rigidity has been explored for other temporal formalisms before [6,7].

For this purpose, we assume in this section that there is a set $\mathsf{N_{RC}} \subseteq \mathsf{N_C}$ of *rigid concept names*. In this setting, a finite sequence $\mathfrak{I} = (\mathcal{I}_i)_{0\leq i\leq n}$ can only be a model of a TKB $\mathcal{K}$ if it fulfills the conditions of Definition 3 and additionally

respects the rigid concept names, i.e. it satisfies $A^{\mathcal{I}_i} = A^{\mathcal{I}_j}$ for every rigid concept name A and all indices i, j between 0 and n.

For the remainder of this section, we restrict the query language to only allow so-called rooted CQs [19]. Intuitively, these are CQs that refer to at least one named individual.

Definition 17. *A CQ ϕ is called* rooted *if (i) it contains at least one free variable or individual name, and (ii) it is* connected*, i.e. for all $x, y \in \mathsf{Var}(\phi) \cup \mathsf{Ind}(\phi)$ there is a sequence $x_1, \ldots, x_n \in \mathsf{Var}(\phi) \cup \mathsf{Ind}(\phi)$ such that $x_1 = x$, $x_n = y$, and for all i, $1 \leq i \leq n$, there is an $r \in \mathsf{N_R}$ such that either $r(x_i, x_{i+1}) \in \mathsf{At}(\phi)$ or $r(x_{i+1}, x_i) \in \mathsf{At}(\phi)$. A TCQ is* rooted *if it contains only rooted CQs.*

This makes sense from an application point of view since one usually does not ask if there is some object with certain properties, but actually wants to know the names of all objects with these properties. This restriction is not without loss of generality, but it is needed in the proof of Lemma 19. We have so far not been able to treat non-rooted TCQs in the presence of rigid concept names.

If we take the approach mentioned in Section 3 of viewing the input ABoxes as a temporal database and rewriting the TCQ into an ATSQL-query as in [14], then the additional rigidity constraints can simply be enforced by triggers that ensure that new knowledge about rigid names is added to the database at all previous time points.

However, the presence of rigid names poses a bigger problem for the incremental algorithm of [13] and that described in Section 5, both of which do not retain the data for all previous time points. For example, if the ABox at the next time point includes the assertion $A(a)$, where A is rigid, then this retroactively also changes the answers to the query $A(x)$ at previous time points. But the aforementioned algorithms assume that the answers at previous time points do not change.

Before we consider how to modify the algorithms for temporal query answering over databases, we have to show that we can still employ the rewriting approach and answer atemporal queries over a TKB $\mathcal{K}$ by directly querying the database $\mathsf{DB}(\mathcal{K})$. This means that we have to reconsider the proof of Theorem 9 regarding the interpretation of rigid names. The main problem we have to solve is that the sequence $\mathfrak{I}_\mathcal{K}$ of canonical models does not necessarily respect the rigid concept names. In the following, let $\mathcal{K} = \langle (\mathcal{A}_i)_{i \geq 0}, \mathcal{T} \rangle$ be an infinite TKB. Similar to Section 5, we denote by $\mathcal{K}^{(n)} := \langle (\mathcal{A}_i)_{0 \leq i \leq n}, \mathcal{T} \rangle$ the finite prefix of $\mathcal{K}$ of length $n+1$. We show how to construct modified sequences of interpretations (similar to $\mathfrak{I}_{\mathcal{K}^{(n)}}$ from Theorem 9) that respect rigid names.

The first step is to find a set $\mathcal{R} \subseteq \{A(a) \mid A \in \mathsf{N_{RC}}, a \in \mathsf{N_I}\}$ that specifies the rigid concept names that the individual names are allowed to satisfy. Of course, we have to ensure that the assertions in $\mathcal{R}$ are not contradicted by any of the ABoxes $\mathcal{A}_i$, $i \geq 0$. We construct $\mathcal{R}$ iteratively, starting from $\mathcal{R}_0 := \emptyset$, as follows. In each step, we add to $\mathcal{R}_j$, $j \geq 0$, all assertions $A(a)$ with $A \in \mathsf{N_{RC}}$ and $a \in \mathsf{N_I}$ that are implied by $\mathcal{A}_i \cup \mathcal{R}_j$ w.r.t. $\mathcal{T}$ for some $i \geq 0$. This reasoning task is called *instance checking* and can be done in polynomial time in *DL-Lite*$_{core}$ [10]. This results in a new set $\mathcal{R}_{j+1}$. We iterate this process until no new assertions are

added. Since there are only polynomially many assertions of the form $A(a)$ as above, this is possible in polynomial time. We denote by $\mathcal{R}$ the final set computed by this procedure. The next lemma shows that, in order to answer TCQs over $\mathcal{K}^{(n)}$, we can equivalently consider the TKB $\mathcal{K}^{(n)}_{\mathcal{R}} := \langle (\mathcal{A}_i \cup \mathcal{R})_{0 \leq i \leq n}, \mathcal{T} \rangle$.

Lemma 18. *Let ϕ be a TCQ and $\mathcal{K} = \langle (\mathcal{A}_i)_{i \geq 0}, \mathcal{T} \rangle$ be an infinite TKB. Then there is a set $\mathcal{R}$ as above such that, for all i and n with $0 \leq i \leq n$, we have*

$$\mathsf{Cert}(\phi, \mathcal{K}^{(n)}, i) = \mathsf{Cert}(\phi, \mathcal{K}^{(n)}_{\mathcal{R}}, i).$$

Note that, if $\mathcal{R}$ is not consistent w.r.t. $\mathcal{T}$, this means that the TKB $\mathcal{K}$ is not consistent, i.e. there is no model of $\mathcal{K}$ that respects the rigid concept names.

Once we have computed $\mathcal{R}$, we can construct the desired sequence of canonical models that respects the rigid concept names, using an idea from [6]. We start with the original sequence $\mathfrak{I}_{\mathcal{K}^{(n)}_{\mathcal{R}}} = (\mathcal{I}_{\mathcal{A}_i \cup \mathcal{R}, \mathcal{T}})_{0 \leq i \leq n}$ that was used in Theorem 9 (but now with $\mathcal{K}^{(n)}_{\mathcal{R}}$ instead of $\mathcal{K}^{(n)}$). It is important to note that these canonical models, as constructed in [10], are all countable. We define the set $\mathcal{D} \subseteq 2^{\mathsf{N_{RC}}}$ of subsets of $\mathsf{N_{RC}}$ that contains exactly the sets

$$\rho(\mathcal{I}_{\mathcal{A}_i \cup \mathcal{R}, \mathcal{T}}, x) := \{A \in \mathsf{N_{RC}} \mid x \in A^{\mathcal{I}_{\mathcal{A}_i \cup \mathcal{R}, \mathcal{T}}}\}$$

for all i, $0 \leq i \leq n$, and $x \in \Delta^{\mathcal{I}_{\mathcal{A}_i \cup \mathcal{R}, \mathcal{T}}}$. We will now modify each $\mathcal{I}_{\mathcal{A}_i \cup \mathcal{R}, \mathcal{T}}$ into a new interpretation $\mathcal{I}_i$ such that for each $Y \in \mathcal{D}$ there are countably infinitely many individuals $x \in \Delta^{\mathcal{I}_i}$ with $Y = \rho(\mathcal{I}_i, x)$.

To this end, consider i, n, $0 \leq i \leq n$, and $Y \in \mathcal{D}$. If $\mathcal{I}_{\mathcal{A}_i \cup \mathcal{R}, \mathcal{T}}$ does not contain any such individual, then we first have to add one. Fortunately, from the definition of $\mathcal{D}$ we know that there must be a j, $0 \leq j \leq n$, and $x \in \Delta^{\mathcal{I}_{\mathcal{A}_j \cup \mathcal{R}, \mathcal{T}}}$ such that $Y = \rho(\mathcal{I}_{\mathcal{A}_j \cup \mathcal{R}, \mathcal{T}}, x)$. To be on the safe side, we therefore construct the disjoint union $\mathcal{I}'_i$ of all interpretations in $\mathfrak{I}_{\mathcal{K}^{(n)}_{\mathcal{R}}}$. More formally, the domain of $\mathcal{I}'_i$ is the disjoint union of the domains of $\mathcal{I}_{\mathcal{A}_j \cup \mathcal{R}, \mathcal{T}}$, $0 \leq j \leq n$. The concept and role names are interpreted as the (disjoint!) union of the interpretations of these names under all $\mathcal{I}_{\mathcal{A}_j \cup \mathcal{R}, \mathcal{T}}$, while the individual names are interpreted as in $\mathcal{I}_{\mathcal{A}_i \cup \mathcal{R}, \mathcal{T}}$. Note that the components of $\mathcal{I}'_i$ are not connected by any roles. This implies in particular that any homomorphism of a *rooted* CQ into $\mathcal{I}'_i$ must actually be a homomorphism into the original canonical model $\mathcal{I}_{\mathcal{A}_i \cup \mathcal{R}, \mathcal{T}}$. This fact is essential to prove Lemma 19 below (see [9] for details).

To ensure that there are even countably infinitely many such individuals, we now define $\mathcal{I}''_i$ as the countably infinite disjoint union of $\mathcal{I}'_i$ with itself, where again the interpretation of the individual names remains unchanged. Finally, we ensure that all models have the same domain $\Delta := \mathsf{N_I} \cup (\mathcal{D} \times \mathbb{N})$ and interpret the individual names by the same domain elements by applying a bijection between the domain of each $\mathcal{I}''_i$ and Δ. In particular, each $a^{\mathcal{I}''_i}$ for $a \in \mathsf{N_I}$ is simply mapped to a, and every other element $x \in \Delta^{\mathcal{I}''_i}$ is mapped to some $(\rho(\mathcal{I}''_i, x), \ell)$ with $\ell \in \mathbb{N}$. We denote the resulting interpretation by $\mathcal{I}_i$ and define $\mathfrak{I}_{\mathcal{K}^{(n)}, \mathcal{R}} := (\mathcal{I}_i)_{0 \leq i \leq n}$.

Algorithm 1: Compute certain answers to a rooted TCQ w.r.t. rigid names

Input : A rooted TCQ ϕ and an infinite TKB $\mathcal{K} = \langle (\mathcal{A}_i)_{i\geq 0}, \mathcal{T}\rangle$
Output : $\mathsf{Cert}(\phi, \mathcal{K}^{(i)})$ for each $i \geq 0$

for $\mathcal{R} \in \mathfrak{R}$ **do**
  initialize an instance $\mathsf{A}_\mathcal{R}$ of the algorithm of Section 5 with $\phi^\mathcal{T}$
end
for $i \leftarrow 0, 1, \ldots$ **do**
  for $\mathcal{R} \in \mathfrak{R}$ **do**
    run $\mathsf{A}_\mathcal{R}$ on input $\mathsf{DB}(\mathcal{A}_i \cup \mathcal{R})$ to compute $\mathsf{Cert}(\phi, \mathcal{K}^{(i)}_\mathcal{R})$
  end
  output $\bigcap_{\mathcal{R}\in\mathsf{Active}} \mathsf{Cert}(\phi, \mathcal{K}^{(i)}_\mathcal{R})$
end

Lemma 19. *The sequence $\mathfrak{I}_{\mathcal{K}^{(n)},\mathcal{R}}$ is a model of $\mathcal{K}^{(n)}_\mathcal{R}$. Furthermore, for all rooted CQs ϕ and every i, $0 \leq i \leq n$, we have*

$$\mathsf{Ans}(\phi, \mathfrak{I}_{\mathcal{K}^{(n)},\mathcal{R}}, i) = \mathsf{Ans}(\phi, \mathfrak{I}_{\mathcal{K}^{(n)}_\mathcal{R}}, i).$$

We can now finally state the variant of Theorem 9 that can deal with rigid concept names.

Theorem 20. *Let ϕ be a rooted TCQ and $\mathcal{K} = \langle (\mathcal{A}_i)_{i\geq 0}, \mathcal{T}\rangle$ be an infinite TKB. Then there is a set $\mathcal{R}$ as above such that, for all i and n with $0 \leq i \leq n$, we have*

$$\mathsf{Cert}(\phi, \mathcal{K}^{(n)}_\mathcal{R}, i) = \mathsf{Ans}(\phi, \mathfrak{I}_{\mathcal{K}^{(n)},\mathcal{R}}, i) = \mathsf{Ans}(\phi^\mathcal{T}, \mathsf{DB}(\mathcal{K}^{(n)}_\mathcal{R}), i).$$

Note that $\mathsf{DB}(\mathcal{K}^{(n)}_\mathcal{R})$ is independent of the construction of $\mathfrak{I}_{\mathcal{K}^{(n)},\mathcal{R}}$, and we can now simply apply the algorithm of Section 5 to the modified sequence of interpretations $\mathsf{DB}(\mathcal{K}^{(n)}_\mathcal{R})$ instead of $\mathsf{DB}(\mathcal{K}^{(n)})$. More formally, let $\mathfrak{R}$ denote the set of all sets $\mathcal{R}$ of the form described above. Algorithm 1 describes the steps necessary to compute the certain answers to a TCQ in the presence of rigid names.

For each $\mathcal{R} \in \mathfrak{R}$, we start an instance $\mathsf{A}_\mathcal{R}$ of the algorithm presented in Section 5. All these instances are run in parallel, with the only difference between them being that each instance has a fixed set $\mathcal{R}$ of assumptions about the rigid names. In each step, every instance $\mathsf{A}_\mathcal{R}$ computes the certain answers to ϕ relative to $\mathcal{R}$, and the actual set of certain answers to ϕ is then computed by taking the intersection over all these sets. Note that we could in each step terminate those instances $\mathsf{A}_\mathcal{R}$ for which $\mathcal{A}_i \cup \mathcal{R}$ is inconsistent w.r.t. $\mathcal{T}$ since this implies that $\mathcal{K}^{(i)}_\mathcal{R}$ has no models, and thus $\mathsf{Cert}(\phi, \mathcal{K}^{(i)}_\mathcal{R}) = \Delta^{\mathsf{N_V}}$ does not contribute to the computation of the intersection in Algorithm 1. However, we leave out this simple optimization to make the presentation of the algorithm clearer.

Theorem 21. *Given a rooted TCQ ϕ and an infinite TKB $\mathcal{K} = \langle (\mathcal{A}_i)_{i\geq 0}, \mathcal{T}\rangle$, Algorithm 1 outputs $\mathsf{Cert}(\phi, \mathcal{K}^{(i)})$ for each $i \geq 0$.*

We have thus extended the algorithm in Section 5—and by extension also the algorithm described in [13]—to deal with rigid concept names in rooted TCQs.

7 Conclusions

We have introduced the reasoning task of *temporal OBDA* over *DL-Lite* knowledge bases and shown how to reduce this task to answering queries over temporal databases, similar to what was done for the atemporal case [10]. We then presented three approaches to solve the latter problem. The first involves storing the whole history of the database and re-evaluating the query at each time point using a temporal database query language like ATSQL [14].

The second approach works by eliminating the future operators and evaluating the resulting query using the algorithm of [13], which achieves a bounded history encoding. Although independent of the length of the history, this involves a non-elementary blow-up in the size of the query. Then, we presented an algorithm that works directly with the future operators. We showed that the algorithm computes exactly the desired answers, but its space requirements are in general not independent of the length of the history. In future work, we will try to achieve a bounded history encoding for certain classes of TCQs, and compare the performance of all three approaches on temporal databases.

Finally, we also described an approach to extend the proposed algorithm to deal with rigid concept names if only rooted CQs are allowed. We plan to investigate how to adapt the algorithm to deal also with rigid role names.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Artale, A., Kontchakov, R., Ryzhikov, V., Zakharyaschev, M.: *DL-Lite* with temporalised concepts, rigid axioms and roles. In: Ghilardi, S., Sebastiani, R. (eds.) Proc. of the 6th Int. Symp. on Frontiers of Combining Systems (FroCoS'09). Lecture Notes in Computer Science, vol. 5749, pp. 133–148. Springer-Verlag (2009)
3. Artale, A., Kontchakov, R., Ryzhikov, V., Zakharyaschev, M.: Temporal conceptual modelling with DL-Lite. In: Proc. of the 2010 Int. Workshop on Description Logics (DL'10). CEUR Workshop Proceedings, vol. 573. CEUR-WS.org (2010)
4. Artale, A., Kontchakov, R., Ryzhikov, V., Zakharyaschev, M.: A cookbook for temporal conceptual data modelling with description logics. CoRR abs/1209.5571 (2012), `http://arxiv.org/abs/1209.5571`
5. Artale, A., Kontchakov, R., Wolter, F., Zakharyaschev, M.: Temporal description logic for ontology-based data access. In: Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI'13). AAAI Press (2013)
6. Baader, F., Borgwardt, S., Lippmann, M.: Temporalizing ontology-based data access. In: Bonacina, M.P. (ed.) Proc. of the 24th Int. Conf. on Automated Deduction (CADE'13). Lecture Notes in Artificial Intelligence, vol. 7898, pp. 330–344. Springer-Verlag (2013)
7. Baader, F., Ghilardi, S., Lutz, C.: LTL over description logic axioms. ACM Transactions on Computational Logic 13(3), 21:1–21:32 (2012)

8. Borgwardt, S., Lippmann, M., Thost, V.: Temporal query answering in *DL-Lite*. In: Proc. of the 26th Int. Workshop on Description Logics (DL'13). CEUR Workshop Proceedings, CEUR-WS.org (2013)
9. Borgwardt, S., Lippmann, M., Thost, V.: Temporal query answering w.r.t. *DL-Lite*-ontologies. LTCS-Report 13-05, Chair of Automata Theory, TU Dresden, Dresden, Germany (2013), see http://lat.inf.tu-dresden.de/research/reports.html.
10. 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.A. (eds.) Reasoning Web, 5th Int. Summer School 2009, Tutorial Lectures, Lecture Notes in Computer Science, vol. 5689, pp. 255–356. Springer-Verlag (2009)
11. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: DL-Lite: Tractable description logics for ontologies. In: Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI'05). pp. 602–607. AAAI Press (2005)
12. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: Hopcroft, J.E., Friedman, E.P., Harrison, M.A. (eds.) Proc. of the 9th Annual ACM Symp. on Theory of Computing (STOC'77). pp. 77–90. ACM Press (1977)
13. Chomicki, J.: Efficient checking of temporal integrity constraints using bounded history encoding. ACM Transactions on Database Systems 20(2), 148–186 (1995)
14. Chomicki, J., Toman, D., Böhlen, M.H.: Querying ATSQL databases with temporal logic. ACM Transactions on Database Systems 26(2), 145–178 (2001)
15. Gabbay, D.: Declarative past and imperative future. In: Banieqbal, B., Barringer, H., Pnueli, A. (eds.) Proc. of the 1987 Coll. on Temporal Logic in Specification. Lecture Notes in Computer Science, vol. 398, pp. 409–448. Springer-Verlag (1989)
16. Gutiérrez-Basulto, V., Klarman, S.: Towards a unifying approach to representing and querying temporal data in description logics. In: Krötzsch, M., Straccia, U. (eds.) Proc. of the 6th Int. Conf. on Web Reasoning and Rule Systems (RR'12). Lecture Notes in Computer Science, vol. 7497, pp. 90–105. Springer-Verlag (2012)
17. Hülsmann, K., Saake, G.: Theoretical foundations of handling large substitution sets in temporal integrity monitoring. Acta Informatica 28(4), 365–407 (1991)
18. Laroussinie, F., Markey, N., Schnoebelen, P.: Temporal logic with forgettable past. In: Proc. of the 17th Annual IEEE Symp. on Logic in Computer Science (LICS'02). pp. 383–392. IEEE Press (2002)
19. Lutz, C.: The complexity of conjunctive query answering in expressive description logics. In: Proc. of the 4th Int. Joint Conf. on Automated Reasoning (IJCAR'08). Lecture Notes in Artificial Intelligence, vol. 5195, pp. 179–193. Springer-Verlag (2008)
20. Pnueli, A.: The temporal logic of programs. In: Proc. of the 18th Annual Symp. on Foundations of Computer Science (SFCS'77). pp. 46–57 (1977)
21. Saake, G., Lipeck, U.W.: Using finite-linear temporal logic for specifying database dynamics. In: Börger, E., Büning, H.K., Richter, M.M. (eds.) Proc. of the 2nd Workshop on Computer Science Logic (CSL'88), Lecture Notes in Computer Science, vol. 385, pp. 288–300. Springer-Verlag (1989)
22. Toman, D.: Logical data expiration. In: Chomicki, J., van der Meyden, R., Saake, G. (eds.) Logics for Emerging Applications of Databases, chap. 6, pp. 203–238. Springer-Verlag (2004)
23. Wilke, T.: Classifying discrete temporal properties. In: Proc. of the 16th Annual Symp. on Theoretical Aspects of Computer Science (STACS'99). Lecture Notes in Computer Science, vol. 1563, pp. 32–46. Springer-Verlag (1999)