Technische Universität Hamburg-Harburg

Project

Implementation of an Algorithm for Temporal Query Answering on Ontologies

Jan Holste

April 11, 2014

supervised by:
Prof. Dr. Ralf Möller
Dr. Özgür Özcep

Hamburg University of Technology
Institute for Software Systems
Schwarzenbergstrae 95
21073 Hamburg

Eidesstattliche Erklärung

Ich versichere an Eides statt, dass ich die vorliegende Projektarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit wurde in dieser oder ähnlicher Form noch keiner Prüfungskommission vorgelegt.

Hamburg, den _____

Jan Holste

Contents

1. Introduction

Nowadays, a lot of technical equipment is monitored. Sensors are used to record the current behavior of the equipment and it's environment.

One of the goals is to understand the current behavior and to realise upcoming events like erosion, which might end-up in a break down of the equipment. It's important to know, which part of the equipment has to be replaced in the near future. To understand the behavior, a lot of sensor data is stored and analysed. With this knowledge one can create pattern, which describe and identify such behavior. The pattern are queries over the set of sensor data.

With the help of ontology-based data access (OBDA), query answering in relational databases can be generalised. It is based on Ontologies, which abstracts the relations of the database. With the description logic DL-Lite, queries with Ontologies are described. They are queried against the database by rewriting the query. The rewriting of a query is the transformation into a first-order query by compiling the information of the Ontologies into the query.

The sensor data are temporal and represented as a sequence of sensor data. The algorithm [2] is based on a temporal version of OBDA and uses the operators of the temporal logic LTL to recognise situations, which are evolve over time.

The first chapter deals with the fundamentals of DL-Lite and the temporal conjunctive queries. The second part is the algorithm [2]. The main part of this project is the implementation in Chapter 3. At the end i discuss the correctness and future work.

2. Fundamentals

In this chapter, i present the fundamentals [2] of DL-Lite, temporal conjunctive queries and the algorithm. My implementation of this algorithm is described in Chapter 3.

2.1. DL-Lite

To achieve query answering on the basis of database systems, i need a description logic, which can be built on top of classical database system to retrieve the data. The relations of the relational database are abstracted by using Ontologies.

The approach to use DL-Lite to query databases with ontology-based data access is done in [3]. The algorithm presented in [1] uses this approach to query CQs against the database.

Syntax In DL-Lite, a concept is represented by a symbol in N_C, a role by a symbol in N_R and an individual name by a symbol in N_I. All sets are not empty and pairwise disjoint. A basic concept has the form $A \in N_C$ or $\exists R \in N_R$. A general concept is B or $\neg B$, where B represents a basic concept. A role expression can be a role name $P_1 \in N_R$ or an inverse role $P_1^- = P_2$ with $P_2 \in N_R$.

The A-Box is a set of assertions. Assertions have the form $A(a)$ or $P(a, b)$, where A is a concept ($A \in N_C$), a and b individual names ($a, b \in N_I$) and P is a role ($P \in N_R$). The T-Box is a set of concept inclusions, also called ontology. A concept inclusion is of the form $B \sqsubseteq C$, where C is general concepts and B is a basic concept.

Semantics An interpretation is defined as a pair $I = (\Delta^I, \cdot^I)$. The Δ^I is called domain and is non-empty. The $\cdot^I$ is an assignment function that assigns to every concept name $A \in N_C$ a set $A^I \subseteq \Delta^I$, for every role $P \in N_R$ a set of binary relations $P_I \subseteq \Delta^I \times \Delta^I$ and for every individual $a \in N_I$ an interpretation of a as $a^I \in \Delta^I$. The interpretations of expressions and concepts are as follows:

- $(P^-)^I := \{(e, d) | (d, e) \in P^I\}$
- $(\exists R)^I := \{d | \exists e \in \Delta^I \wedge (d, e) \in P^I\}$

- $(\neg C)^I := \Delta^I \setminus C^I$

The interpretations I is a model of an axiom α iff:

- $B^I \subseteq C^I$ if $\alpha = B \sqsubseteq C$
- $a^I \in A^I$ if $\alpha = A(a)$
- $(a^I, b^I) \in P^I$ if $\alpha = P(a, b)$

$I \models \alpha$ is true if I is a model of the axiom α. $I \models A$ is true, if I is an interpretation of all assertions of the A-Box A. $I \models T$ is true, if I is an interpretation of all concept inclusions in the T-Box. If and only if there is an interpretation I of the A-Box and the T-Box, than its union is consistent. Further more, i assume that the interpretations have unique names in terms of individual names. For two individual names, i get two different interpretations.

2.2. Temporal Conjunctive Queries

To answer temporal conjunctive queries, [2] introduces the notation of a temporal knowledge base $K = \langle (A_i)_{0 \leq i \leq n}, T \rangle$. This notation describes a knowledge base with a finite sequence of n A-Boxes and one T-Box, which is constant over time. Concept names of the A-Boxes have to occur in the T-Box.

Let $I_i = (\Delta, \cdot^{I_i})$ be an interpretation of an A-Box A_i in the sequence of A-Boxes and $J = (I)_{0 \leq i \leq n}$ be a sequence of interpretations. If Δ is a fixed and non-empty domain, then $J \models K$ if and only if for every i both $I_i \models A_i$ and $I_i \models T$ holds.

Syntax A conjunctive query (CQ) is a query of the form $\phi = \exists y_1 \dots y_m . \psi$, where $y_1 \dots y_m \in N_V$ and ψ is a conjunction of atoms. This atoms are either represented by an assertion of $A(z)$, where $A \in N_C$ and $z \in N_V \cup N_I$, or by a role $r(z_1, z_2)$, where $r \in N_R$ and $z_1, z_2 \in N_V \cup N_I$.

By introducing the temporal conjunctive queries (TCQ), i use the linear temporal logic (LTL) specific operators and combine them with the CQs.

Every CQ is also a TCQ. I denote the set of all variables of a TCQ ϕ as $Var(\phi)$. $FVar(\phi)$ denotes the set of all free variables. A free variable is a placeholder for substitution without a specific value. It is bounded, if a specific value is assigned to it. If the set of free variables for a TCQ is empty, the TCQ is called boolean. Every TCQ can form another TCQ with the help of an LTL operator. Iff ϕ_1 and ϕ_2 are TCQs, then also the result of the following operations is a TCQ:

- $\phi_1 \wedge \phi_2$ (conjunction)

- $\phi_1 \vee \phi_2$ (disjunction)

- $\circ\phi_1$ (strong next), $\bullet\phi_1$ (weak next)

- $\circ^-\phi_1$ (strong previous), $\bullet^-\phi_1$ (weak previous)

- $\Box\phi_1$ (always), $\diamond\phi_1$ (eventually)

- $\Box^-\phi_1$ (always in the past), $\diamond^-\phi_1$ (history)

- $\phi_1 U \phi_2$ (since), $\phi_1 S \phi_2$ (until)

Semantics An answer of a TCQ w.r.t. a sequence of interpretations is a mapping of the variables into the domain Δ. To introduce the semantics, i define ψ to be a boolean CQ. A boolean CQ is a query with a boolean result. It bounds all free variables with the exists operator. Let $I = (\Delta, \cdot^I)$ be an interpretation, then a map $\pi : Var(\psi) \cup N_I \to \Delta$ of ψ is a homomorphism iff:

- $\pi(a) = a^I$ for all $a \in N_I$
- $\pi(z) \in A^I$ for all $A(z) \in \psi$
- $(\pi(z_1), \pi(z_2)) \in r^I$ for all $r(z_1, z_2) \in \psi$

I is a model for the boolean CQ ψ, if there is such a mapping π.

To introduce the semantics of TCQ, let ϕ be a boolean TCQ. Let $J = (I_i)_{0 \leq i \leq n}$ a sequence of interpretations. Then $J, i \models \phi$ is defined by induction as follows:

- $J, i \models \exists y_1 \dots y_m \cdot \psi$ iff $I_i \models \exists y_1 \dots y_m \cdot \psi$

- $J, i \models \phi_1 \wedge \phi_2$ iff $J, i \models \phi_1$ and $J, i \models \phi_2$

- $J, i \models \phi_1 \vee \phi_2$ iff $J, i \models \phi_1$ or $J, i \models \phi_2$

- $J, i \models \circ\phi_1$ iff $i < n$ and $J, i + 1 \models \phi_1$

- $J, i \models \bullet\phi_1$ iff $i < n$ implies $J, i + 1 \models \phi_1$

- $J, i \models \circ^-\phi_1$ iff $i > 0$ and $J, i - 1 \models \phi_1$

- $J, i \models \bullet^-\phi_1$ iff $i > 0$ implies $J, i - 1 \models \phi_1$

- $J, i \models \Box\phi_1$ iff for all k with $i \leq k \leq n$ i have $J, k \models \phi_1$

- $J, i \models \diamond\phi_1$ iff there is a k such that $i \leq k \leq n$ and $J, k \models \phi_1$
- $J, i \models \Box^{-}\phi_1$ iff for all k, $0 \leq k \leq i$ i have $J, k \models \phi_1$
- $J, i \models \diamond^{-}\phi_1$ iff there is some k, $0 \leq k \leq i$ such that $J, k \models \phi_1$
- $J, i \models \phi_1 U \phi_2$ iff $\exists k, i \leq k \leq n$ such that $J, k \models \phi_2$ and $J, j \models \phi_1$ for all $j, i \leq j < k$
- $J, i \models \phi_1 S \phi_2$ iff $\exists k, 0 \leq k \leq i$ such that $J, k \models \phi_2$ and $J, j \models \phi_1$ for all $j, k < j \leq i$

In TCQ answering i am interested in the last point n. I assume that there is no point in time before 0 and after n. An interpretation J is a model of ϕ w.r.t. a TKB K, if $J \models K$ and $J, n \models \phi$. If such a model exits, ϕ is also called satisfiable.

The main goal is to find a mapping $a : FVar(\phi) \rightarrow N_I$, where ϕ is a TCQ and $FVar$ represents the set of all free variables in ϕ. This mapping is an answer at a point in time i if $J, i \models a(\phi)$. Furthermore let $K = \langle (A_i)_{0 \leq i \leq n}, T \rangle$ be a temporal knowledge base, then the mapping $a(\phi)$ is an answer for ϕ, if for every $J \models K$, i have $J, i \models a(\phi)$ for the specific point in time i.

The answering of a TCQ ϕ w.r.t. a knowledge base K is accomplished by answering a compiled TCQ ϕ_T over the sequence of interpretations $\langle DB(A_i) \rangle_{0 \leq i \leq n}$. To express the database as sequence of A-Boxes, the interpretation $DB(A) := (N_I, \cdot^{DB(A)})$ is defined as follows:

- $a^{DB(A)} := a$ for all $a \in N_I$
- $A^{DB(A)} := \{a | A(a) \in A\}$ for all $A \in N_C$
- $P^{DB(A)} := \{(a, b) | P(a, b) \in A\}$ for all $P \in N_R$

2.3. The Algorithm

In this subsection, i want to present the new algorithm of [2], which calculates the set of answer formulas recursively without future operator elimination. The algorithm outputs the mapping $\Phi : Sub(\psi) \rightarrow AF^i_\phi$, where $Sub(\psi)$ is the set of all subformulas in ψ and AF^i_ϕ is the set of all answer formulas of ϕ at the point in time i. The algorithm has four main parts. The first part is the initial answer formula, which sets a start point for the algorithm. The second part is the next answer formula, which is based on the initial answer formula and calculates the answer for the next point in time. A few answers can contain placeholder for answer formulas of future operators

(strong next, weak next, until). Therefore i have to update answer formulas, contain placeholders. This is done by the third function, the update formula. To evaluate a query to a given point in time i, i have to return the set of all answer formulas. As mentioned before, i am interested in the last point in time. The answer formulas at the last point in time can contain placeholders, too. Therefore the evaluation function eliminates all occurrences of placeholders, before returning the result.

2.3.1. The Initial Answer Formula

I present the initial answer formulas in this subsection. The term $Ans(\phi, J, i)$ represents the set of all mappings at the point in time i with the interpretation J of the TCQ ϕ. Another way to define this is $Ans(\phi, J^{(i)})$. For the future operations, i introduce a placeholder variable x_i^{ψ}. The ψ denotes the subformula and the i the i-th A-Box A_i, in which this placeholder has to be evaluated. The initial answer formula is a function Φ_0, where the 0 is the first A-Box in the sequence. It is evaluated over all subformulas of the temporal conjunctive query. The set of subformulas contains the temporal conjunctive query, too. The definition of the next answer formula is:

- $\Phi_0(\psi_1) := Ans(\psi_1, J^{(0)})$ if ψ_1 is a CQ
- $\Phi_0(\psi_1 \wedge \psi_2) := \Phi_0(\psi_1) \cap \Phi_0(\psi_2)$
- $\Phi_0(\psi_1 \vee \psi_2) := \Phi_0(\psi_1) \cup \Phi_0(\psi_2)$
- $\Phi_0(\circ\psi_1) := x_0^{\circ\psi_1}$, $\Phi_0(\circ^-\psi_1) := \emptyset$
- $\Phi_0(\bullet\psi_1) := x_0^{\bullet\psi_1}$, $\Phi_0(\bullet^-\psi_1) := \Delta^{N_V}$
- $\Phi_0(\psi_1 U \psi_2) := \Phi_0(\psi_2) \cup (\Phi_0(\psi_1) \cap x_0^{\psi_1 U \psi_2})$
- $\Phi_0(\psi_1 S \psi_2) := \Phi_0(\psi_2)$

I define the initial set of the strong previous operator ($\circ^-$) to be empty, because the previous point in time $i-1$ for $i = 0$ is not in the knowledge base $K = \langle (A_i)_{0 \leq i \leq n}, T \rangle$. The set of the weak previous operator ($\bullet^-$) is the set of all variables of the domain Δ^{N_V}, because $i > 0 \implies J, i-1 \models \phi_1$ holds and $i > 0$ is false.

This definition already returns answer formulas to queries, which don't use the future operators. If the query uses future operators, the initial answer function returns answer formulas, which include placeholder variables of the form x_0^{ψ}. The subformula ψ of the placeholder can be one of the following formulas:

- $\psi = \circ\psi_1$ (strong next)

- $\psi = \bullet\psi_1$ (weak next)

- $\psi = \psi_1 U \psi_2$ (until)

I introduce the update formula later, to deal with the placeholder variables for future operators.

2.3.2. The Next Answer Formula

In the previous subsection i defined the initial answer formula to calculate the answer formulas for the initial point in time. I modify this definition to compute the next set of answer formulas at a point in time $i > 0$. The next answer formula uses the previous calculated answer formulas for the past operators (strong previous, weak previous and since). An answer formula is denoted by Φ_i^0, where i is the current point in time and 0 denotes the current level of the sub formula.

The next answer formula is defined as:

- $\Phi_i^0(\psi_1) := Ans(\psi_1, J^{(i)})$ if ψ_1 is a CQ
- $\Phi_i^0(\psi_1 \wedge \psi_2) := \Phi_i^0(\psi_1) \cap \Phi_i^0(\psi_2)$
- $\Phi_i^0(\psi_1 \vee \psi_2) := \Phi_i^0(\psi_1) \cup \Phi_i^0(\psi_2)$
- $\Phi_i^0(\circ\psi_1) := x_i^{\circ\psi_1}$, $\Phi_i^0(\circ^-\psi_1) := \Phi_{i-1}(\psi_1)$
- $\Phi_i^0(\bullet\psi_1) := x_i^{\bullet\psi_1}$, $\Phi_i^0(\bullet^-\psi_1) := \Phi_{i-1}(\psi_1)$
- $\Phi_i^0(\psi_1 U \psi_2) := \Phi_i^0(\psi_2) \cup (\Phi_i^0(\psi_1) \cap x_i^{\psi_1 U \psi_2})$
- $\Phi_i^0(\psi_1 S \psi_2) := \Phi_i^0(\psi_2) \cup (\Phi_i^0(\psi_1) \cap \Phi_{i-1}(\psi_1 S \psi_2))$

The next answer formula uses Φ_{i-1} to access the previous answer formulas. With this extension, i can answer queries over a sequence of a knowledge base $K = \langle (A_i)_{0 \leq i \leq n}, T \rangle$, if the TCQ does not contain any future operators. If the TCQ contains future operators, the answer formulas contain placeholder variables. To answer queries at the last point in time, the evaluation function has to eliminate the placeholder of the last point in time.

For intermediate answer formulas, i introduce the update formula in the next subsection. It replace placeholder of the previous point in time with the answer formulas calculated by the next answer formula of the current point in time.

2.3.3. The Update Formula

After every execution of the next answer formula, i call the update formula to replace the placeholder of the previous point in time. The placeholder of the previous point in time have the form x_{i-1}^{ψ}. I replace the placeholder with the answer formula of ψ at the current point in time. I define the replacement as:

$$update(x_{i-1}^{\psi^j}) := \begin{cases} \Phi_i^{j-1}(\psi_1) & if \psi^j = \circ\psi_1 \text{ or } \psi^j = \bullet\psi_1 \\ \Phi_i^{j-1}(\psi_j) & if \psi^j = \psi_1 U \psi_2 \end{cases}$$

In the update formula, the $\Phi_i^{j-1}(\psi_1)$ is the replacement for strong next and weak next. The replacement are answer formulas at point in time i of the set of answer formulas $j-1$, where $j-1$ is an intermediate result. Previously, i introduced an order on the set of sub formulas. This order determines the update sequence of subformulas. The first update at point in time 1 would be on the result of the first run of the next answer formula. This is $\Phi_0(\psi) = \Phi_1^0(\psi)$. The update of the first sub formula results in $\Phi_1^1(\psi)$. After updating all subformulas, i arrive at $\Phi_1^j(\psi) = \Phi_1^k(\psi) = \Phi_1(\psi)$, where k is the total number of subformulas and this is the final answer formula set for point in time 1. To consume the next point in time, the next answer formula and the update formula have to be repeated. To evaluate the result, the evaluation function has to use the current answer formula set Φ_1.

2.3.4. Evaluation Function

For evaluation of answer formulas at a given point in time, i introduce the evaluation function. The evaluation function $eval^n : AF_\phi^n \to 2^{\Delta^{N_V}}$ is defined as:

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

This function maps the set of all answer formulas to the set of answer formulas, which represents a solution without placeholder, union and intersection. AF_ϕ^i is the smallest set of answer formulas that satisfy the following conditions:

- Every set $A \subseteq \Delta^{N_V}$ is an answer formula for ϕ at point in time i
- Every placeholder variable $x_j^{\psi} \in Var_j^{\phi}$ with $j \leq i$ is an answer formula for ϕ at point in time i
- Every intersection or union of two answer formulas results into a new answer formula

The evaluation of placeholder variables has four different cases. If i am not at the end of the sequence of A-Boxes, i can evaluate the future operators to a specific set. Therefore i replace the placeholder variable with the answer formula of the next point in time. In case of the next operator, this would be the sub formula ϕ_1. Otherwise in terms of the until operator, i replace it with the answer formula of the next point in time with the same sub formula ϕ.

If the last point in time of the sequence of A-Boxes is arrived, i distinguish between two different cases. The first one are the strong future operators strong next and until. They are evaluated to the empty set of answer formulas. The other case is the weak next. It will be evaluated to the set of all answer formulas.

2.3.5. The Complete Algorithm

In this subsection, i put the whole algorithm together. As mentioned before, i have a loop in the algorithm (Algorithm 1), which repeats until the end of the sequence is arrived.

Algorithm 1 The new algorithm of [1]

Input: A TCQ ϕ and an infinite sequence $J = (I_i)_{i \geq 0}$ of interpretations
Output: $Ans(\phi, J^{(i)})$ for $i \leq 0$

```
for i ← 0, 1 ... do
  if i=0 then
    compute Φ_0; //The initial answer formula
  else
    compute Φ_i^0 from Φ_{i−1}; //The next answer formula
    compute Φ_i^1, ..., Φ_i^k = Φ_i; //The update formula
  end if
end for
output eval^i(Φ_i(φ));
```

This algorithm outputs the result of the TCQ at the end of the loop. In our implementation, i will have a finite sequence of A-Boxes and therefore always reach the end.

2.3.6. Example

This example is based on a sequence of 3 A-Boxes. Each A-Box has predicates $hasValue$ with sensors S as subjects and values of type double as objects. The A-Boxes are:

- $A_0 = \{hasValue(S_0, 2.0), hasValue(S_1, 3.0)\}$
- $A_1 = \{hasValue(S_0, 2.4), hasValue(S_1, 2.9)\}$
- $A_2 = \{hasValue(S_0, 2.2), hasValue(S_1, 3.1)\}$

The TCQ $SP(SP(hasValue(x, y) \wedge SN(hasValue(x, z))))$ queries all sensors with their values of the two previous points in time. The strong previous operator is denoted by SP. The variable x represents a variable for subjects. The variables y, z represents objects. The query has following subformulas:

- $v_0 = hasValue(x, y)$
- $v_1 = hasValue(x, z)$
- $v_2 = SN(v_1)$
- $v_3 = v_0 \wedge v_2$
- $v_4 = SP(v_3)$
- $v_5 = SP(v_4)$

The result of the initial answer formula is:

- $\Phi_0^0(v_0) = \{\{x \to S_0, y \to 2.0\}, \{x \to S_1, y \to 3.0\}\}$
- $\Phi_0^1(v_1) = \{\{x \to S_0, z \to 2.0\}, \{x \to S_1, z \to 3.0\}\}$
- $\Phi_0^2(v_2) = x_0^{\circ v_1}$
- $\Phi_0^3(v_3) = \{\{x \to S_0, y \to 2.0\}, \{x \to S_1, y \to 3.0\}\} \cap x_0^{\circ v_1}$
- $\Phi_0^4(v_4) = \emptyset$
- $\Phi_0^5(v_5) = \emptyset$

The $x_0^{\circ v_1}$ denotes the placeholder variable for the answer formula of v_1 in the next point of time. The result of the next answer formula is:

- $\Phi_1^0(v_0) = \{\{x \to S_0, y \to 2.4\}, \{x \to S_1, y \to 2.9\}\}$
- $\Phi_1^1(v_1) = \{\{x \to S_0, z \to 2.4\}, \{x \to S_1, z \to 2.9\}\}$
- $\Phi_1^2(v_2) = x_1^{\circ v_1}$
- $\Phi_1^3(v_3) = \{\{x \to S_0, y \to 2.4\}, \{x \to S_1, y \to 2.9\}\} \cap x_1^{\circ v_1}$
- $\Phi_1^4(v_4) = \Phi_0^3(v_3) = \{\{x \to S_0, y \to 2.0\}, \{x \to S_1, y \to 3.0\}\} \cap x_0^{\circ v_1}$
- $\Phi_1^5(v_5) = \emptyset$

The result of the update formula of the point in time $i = 1$ is:

- $\Phi_1^4(v_4) = \Phi_0^3(v_3) = \{\{x \to S_0, y \to 2.0, z \to 2.4\}, \{x \to S_1, y \to 3.0, z \to 2.9\}\}$

The result of the next answer formula of the point in time $i = 2$ is:

- $\Phi_2^0(v_0) = \{\{x \to S_0, y \to 2.2\}, \{x \to S_1, y \to 3.1\}\}$
- $\Phi_2^1(v_1) = \{\{x \to S_0, z \to 2.2\}, \{x \to S_1, z \to 3.1\}\}$
- $\Phi_2^2(v_2) = x_2^{\circ v_1}$
- $\Phi_2^3(v_3) = \{\{x \to S_0, y \to 2.2\}, \{x \to S_1, y \to 3.1\}\} \cap x_2^{\circ v_2}$
- $\Phi_2^4(v_4) = \Phi_1^3(v_3) = \{\{x \to S_0, y \to 2.4\}, \{x \to S_1, y \to 2.9\}\} \cap x_1^{\circ v_1}$
- $\Phi_2^5(v_5) = \Phi_1^4(v_4) = \{\{x \to S_0, y \to 2.0, z \to 2.4\}, \{x \to S_1, y \to 3.0, z \to 2.9\}\}$

The result of the update formula of the point in time $i = 2$ is:

- $\Phi_2^4(v_4) = \Phi_1^3(v_3) = \{\{x \to S_0, y \to 2.4, z \to 2.2\}, \{x \to S_1, y \to 2.9, z \to 3.1\}\}$

The complete sequence is consumed. The next step is the evaluation. The evaluation function evaluates the subformula v_5 of point in time $i = 2$. The result of the query is:

$$\{\{x \to S_0, y \to 2.0, z \to 2.4\}, \{x \to S_1, y \to 3.0, z \to 2.9\}\}$$

3. Implementation

In this chapter i describe the implementation of the whole algorithm. The first step is the implementation of the set of answer formulas. I built a query parser to parse TCQs with the help of antlr [4], which is a common tool to generate parsers with the help of a grammar. If both basic parts are implemented, i start to implement the whole algorithm with its helper functions.

3.1. Answer formulas

$Ans(\phi, J, i)$ denotes the set of answer formulas, where J represents the sequence of interpretations of the TCQ ϕ and i represents the point in time. An answer formula is a certain answer, an union, an intersection or a placeholder variable. A certain answer is a mapping $a : FVar(\phi) \to N_I$, where $FVar$ are all free variables of the TCQ ϕ and N_I are the individual names. The placeholder variables are placeholders for answer formulas. They represents answer formulas of future operators, which are calculated in the next point in time. The union and intersection represents sets of two answer formulas, in which a placeholder variable prevent the simplification to one answer formula. No certain answer for a query exists, if the set of answer formulas is empty. The set of all mappings is Δ^{N_V}. A certain answer represents all mappings with an indicator $\top$. With this indicator, the need of calculating all mappings is eliminated.

3.2. Query Parsing

The query parsing is implemented with the help of antlr [4]. The grammar describes TCQ formulas with the past (strong previous, weak previous, since) and future operators (strong next, weak next, until, box, diamond). A CQ can have concrete domains, exists operators, and assertions. An assertion has a predicate, a subject and an optional object. The subject and the predicate are represented by an URI. The optional object can be an URL or a literal. A literal represents different data types with their values.

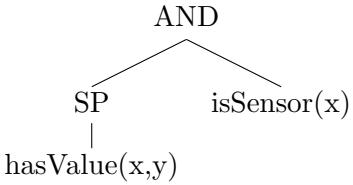

Figure 3.1.: The TCQ $\phi = SP(hasValue(x, y))\ AND\ isSensor(x)$ parsed as tree

The main task for the parser is to bring the query into a tree shaped structure. The implementation of the algorithm functions is based upon this structure. A tree traversal through this tree in post-order delivers an order, which represents always the same order of subformulas. The initial answer formula, the next answer formula, and the update formula uses this post-order traversal to calculate answer formulas of the subformulas first. An example is given in Figure 3.1. The traversal of the TCQ

$$\phi = SP(hasValue(x, y))\ AND\ isSensor(x)$$

in post-order is given by:

$$hasValue(x, y), SP(hasValue(x, y)),$$

$$isSensor(x), SP(hasValue(x, y))\ AND\ isSensor(x)$$

3.3. The Initial Answer Formula

The first function of the algorithm is the initial answer formula. It calculates all answer formulas of a given sequence at the first point in time. The execution is recursive over the structure of the TCQ. The traversal is post-order, therefore every subformula is calculated first. The past operators are calculated to a certain answer, an empty set of answer formulas, or a set with all mappings denoted by $\top$. The result of the weak previous operator in the first point in time is $\top$. The strong previous operator has the empty set of answer formulas as result. All subformulas of the past operators are calculated, too. They are used by the next execution of the next answer formula. The result of future operators are placeholder variables. They are replaced by a specific answer formula in the update formula. The subformulas of the future operators are calculated, too. They can contain past operators and therefore the results are needed later in the next answer formula. The conjunction and disjunction uses the calculated results of the subformulas to represent the result as intersection or union. If the subformulas do not contain any placeholder variables, the answer formulas are simplified to one set of answer formulas.

3.4. The Next Answer Formula

The structure of the implementation of the next answer formula is quite the same as before the initial answer formula. The only change is the implementation of the past operators. The execution of the next answer formula is recursive over the the structure of the TCQ. For the past operators, the result of the previous point in time is used. The operators weak previous and strong previous uses the result of the subformula of the previous point in time. The since operator uses its own result of the previous point in time.

The algorithm uses the previous point in time to calculate the current one. Only the results of two points in time are stored. If the next answer formula is executed, the previous point in time is never used again and can be removed.

3.5. The Update Formula

The update formula is executed after each run of the next answer formula. It update all answer formulas of the current point in time by replacing all placeholder variables of the previous point in time. The placeholder variables are introduced by future operators. All replacements are answer formulas of the current point in time. To ensure, that every placeholder variable of the previous step is replaced, i update the answer formulas in the same order as the post-order traversal of the TCQ. The answer formulas contain placeholder variables, certain answers, intersections and unions. The answer formulas are updated recursively. In this recursion, i simplify answer formulas with intersection and union, which placeholder variables are eliminated. This decrease the memory use and improve the performance, if a TCQ is evaluated at a point in time twice.

A second execution of the update formula or the execution after the initial answer formula do not change the answer formula sets, because no placeholder variables of the previous point in time are present.

3.6. Evaluation Function & Main Function

The algorithm has two main parts. The first part is the consuming of the sequence. If the consuming point in time is the first one, the initial answer formula is called. Otherwise the next answer formula and the update formula are called. The second part of the algorithm is the evaluation function. It takes the answer formulas of the TCQ in the last point in time and replace all occurrences of placeholder variables.

The evaluation function is only performed, if the consuming function is finished. The structure of the evaluation function is the same as the update formula, only the placeholder variables are replaced with different answer formulas. A placeholder variable of weak next or box is replaced by the set of all mappings. The empty set of answer formulas is the replacement for placeholder variables of strong next, until, and diamond.

3.7. The Box and Diamond Operator

The implementation of the Box and Diamond operator has the same structure as that for the Until and Since operator. Furthermore, i implement the history and always in the past operator. I introduce the extension in two parts. The first part shows the syntax and semantics, the other part the algorithm extension.

Syntax The syntax for Box and Diamond is:

- $\Box\phi_1$ (always)
- $\diamond\phi_1$ (eventually)
- $\Box^-\phi_1$ (always in the past)
- $\diamond^-\phi_1$ (history)

Semantics The semantics is:

- $J, i \models \Box\phi_1$ iff for all k, $i \leq k \leq n$ i have $J, k \models \phi_1$
- $J, i \models \diamond\phi_1$ iff there is some k, $i \leq k \leq n$ such that $J, k \models \phi_1$
- $J, i \models \Box^-\phi_1$ iff for all k, $0 \leq k \leq i$ i have $J, k \models \phi_1$
- $J, i \models \diamond^-\phi_1$ iff there is some k, $0 \leq k \leq i$ such that $J, k \models \phi_1$

Algorithm I have the syntax and the semantics of the operators. The extension of the initial answer formula is:

- $\Phi_0(\Box\psi) = \Phi_0(\psi) \cap x_0^{\Box\psi}$
- $\Phi_0(\Diamond\psi) = \Phi_0(\psi) \cup x_0^{\Diamond\psi}$
- $\Phi_0(\Box^-\psi) = \Phi_0(\psi)$
- $\Phi_0(\Diamond^-\psi) = \Phi_0(\psi)$

The extension of the next answer formulas is:

- $\Phi_i(\Box\psi) = \Phi_i(\psi) \cap x_i^{\Box\psi}$
- $\Phi_i(\Diamond\psi) = \Phi_i(\psi) \cup x_i^{\Diamond\psi}$
- $\Phi_i(\Box^-\psi) = \Phi_i(\psi) \cap \Phi_{i-1}(\Box^-\psi)$
- $\Phi_i(\Diamond^-\psi) = \Phi_i(\psi) \cup \Phi_{i-1}(\Diamond^-\psi)$

The update formula is:

$$update(x_{i-1}^{\psi^j}) := \begin{cases} \Phi_i^{j-1}(\psi_1) & if \psi^j = \circ\psi_1 \text{ or } \psi^j = \bullet\psi_1 \\ \Phi_i^{j-1}(\psi^j) & if \psi^j = \psi_1 U \psi_2 \text{ or } \psi^j = \Box\psi_1 \text{ or } \psi^j = \Diamond\psi_1 \end{cases}$$

The extension of the evaluation formula is:

- $eval^n(x_j^{\psi}) := \begin{cases} Ans(\psi, J^{(n)}, j+1) & if\ j < n \ and\ (\psi = \Box\psi_1 \ or\ \psi = \Diamond\psi_1) \\ \emptyset & if\ j = n \ and\ \psi = \Diamond\psi_1 \\ \Delta^{N_V} & if\ j = n \ and\ \psi = \Box\psi_1 \end{cases}$

3.8. Exists Operator

In this subsection, i describe the implementation of the exists operator for CQs. An exists limit the scope of a variable to a conjunctive query. Given the CQ $\phi = \exists x.(A(x, y))$, then only mappings of y, for all records $A(x, y)$ in the database, are visible to the scope of ϕ. The exists operator bounds the x variable. Another variable with the same name outside this scope, is ignored. Given a TCQ $\phi = \Box(A(x, y) \wedge \exists.xB(x))$, then the x of $B(x)$ is limited to the scope of the exists and represents another instance than the x of $A(x, y)$.

The implementation of the exists operator removes every mapping of the bounded variables, which scopes end at the exists operator. If the answer formula is empty

after this operation and there exists a none empty set before, i return an answer formula with all mappings.

The implementation of the exists operator extends the parser grammar. It introduces a syntax for exists with one or more bounded variables. The implementation of the initial answer formula, the next answer formula and the update formula have to handle the recursion of queries, which includes the exists operator. The answer formula of an exists operator is the answer formula of its subformula without mappings of the bounded variables.

3.9. Concrete Domains

In this subsection, i describe the implementation of the concept of concrete domains. With a concrete domain, i can limit the value represented by a variable. Concrete domains are only allowed in CQs. Let $\phi = hasValue(x, y) \wedge y < 10.0$ be a CQ, then $y < 10.0$ is a concrete domain. The variable x represents a sensor subject and y is from data type double, then i get only $assignments$, that contain sensor values less than 10.0.

A comparison between two variables is not allowed, because they can not be evaluated to an interval in time. A CQ with concrete domain is valid, if every bounded variable of the concrete domain is used in the atoms of the CQ. The CQ $A(x) \wedge y > 10.0$ is not valid, because y is not used in $A(x)$.

To handle concrete domains in terms of answer formulas, i extend the answer formulas with two lists of extremum. The first list contains variables and their minimum. The minimum of a variable is $-\infty$, iff the variable is not in the list. The second list is for maximum.

To calculate the intersection or union of two answer formulas, i calculate the new extremum first. Every answer formula is removed, if it has a value for a variable that is outside of the new boundaries of this specific variable.

The union of two intervals is defined as:

$$[a_1, a_2] \cup [b_1, b_2] = [max(a_1, b_1), min(a_2, b_2)]$$

The new extremum for an union of two answer formulas is defined as follows:

$$a.max(x, op_1, v_1) \cup b.max(x, op_2, v_2) = \begin{cases} (\mathrm{x}, \mathrm{op}_1, v_1) & v_1 \ op_1 \ v_2 \\ (\mathrm{x}, \mathrm{op}_2, v_2) & otherwise \end{cases}$$

$$a.min(x, op_1, v_1) \cup b.min(x, op_2, v_2) = \begin{cases} (\text{x,op}_1, v_1) & v_1 \ op_1 \ v_2 \\ (\text{x,op}_2, v_2) & otherwise \end{cases}$$

The intersection of two intervals is defined as:

$$[a_1, a_2] \cup [b_1, b_2] = [max(a_1, b_1), min(a_2, b_2)]$$

The new extremum for an intersection of two answer formulas is defined as follows:

- $a.max(x, op_1, v_1) \cup b.max(x, op_2, v_2) = \begin{cases} (\text{x,op}_2, v_2) & v_1 \ op_1 \ v_2 \\ (\text{x,op}_1, v_1) & otherwise \end{cases}$

- $a.min(x_1, op_1, v_1) \cup b.min(x_2, op_2, v_2) = \begin{cases} (\text{x,op}_2, v_2) & v_1 \ op_1 \ v_2 \\ (\text{x,op}_1, v_1) & otherwise \end{cases}$

The operators op_1, op_2 can be $<, \leq$ for a maximum extrema and $>, \geq$ for a minimum value.

4. Evaluation

I have implemented the algorithm with the Box and Diamond operator, the concept of concrete domains, and the exists operator. In this chapter, i evaluate this new implementation.

I tested the algorithm manually over the test data of Optique from Siemens. The test data is a sequence of A-Boxes, that includes sensor data. Every A-Box has an assertion for each sensor with the predicate $hasValue$. The subject is the sensor and the object is the measurement as *double*.

The tests covers all basic operations and all allowed combinations of two operations. Every transfer of intermediate results between operations is tested with this coverage. Every operator in the initial and next answer formula loads and stores answer formulas. If a test for an operator is correct and the combination with all other operators are correct, then all queries with this operators are correct. In my tests all query results are correct.

The test data has only assertions with objects. Furthermore, the data type of the objects is always double. The correctness of the testing is limited to this facts. A total correctness of the algorithm can be only shown with more data, that contain assertions without objects and assertions with all possible data types, not only double. The testing is done manually. To achieve a more reliability and comfort testing, the testing has to be automated.

The performance of the algorithm relies on the number of subformulas, the length of the consuming sequence of A-Boxes and the size of record mappings for every assertion, which is queried against the database. The initial answer formula is called only once recursively over the set of subformulas. The number of recursive calls of the initial answer formula depends on the size of the query. Each recursive call calculates answer formulas, which may contain placeholder, union and/or intersection. The performance of an intersection or union depends on the size of both sets.

The consuming performance of a sequence depends on its length, the next initial answer formula and the update formula. Both, the next answer and the update formula are called recursive over the subformulas of the query for every A-Box.

Therefore the performance decreases by large queries. Another issue is the simplification of answer formulas by calculating the intersections and unions. Not always can be the set simplified, which can result in a growing allocation of memory and a slow down of set operations.

The evaluation function is a simplification of the current answer formulas of the whole query to answer formulas without placeholder variables. If the answer formulas contain placeholder, which prevents the simplification in the steps before, a large number of intersections and/or unions have to be performed. This can be improved by reordering of the set operations intersection and union. Every consumption of an A-Box should be followed by reordering the set operations.

Let be given a long sequence with an assertion only used in the current point in time. Let the database contain a lot records for this assertion, then every consumption of the A-Boxes in the sequence would query all records of this assertion, but only the last point in time is using this created mappings.

All in all, the performance of the algorithm depends on the query complexity and the size of the A-Box in the sequence of A-Boxes. The manual testing shows the correctness with limit to the test data. To validate the correctness of this algorithm, more test data and also more different data types are needed.

5. Future Work

I implemented an algorithm for DL-Lite Query Answering over A-Box Sequences. This algorithm takes a query and calculates the answer set of the given sequence by breaking it down to assertions, which are queried against a database. This database contains only data mapped to A-Box assertions. Therefore the query can contain only assertions and no ontology rules. To support ontology rules, i have to rewrite the T-Box.

The implementation of the update formula and the next answer formula are both recursive. A merging of both formulas in the implementation can result into a performance improvement, because only one recursion over the query is executed.

There are a few answer formulas, that are computed, but never used. For instance queries, which use future operators. In the initial answer formula, subformulas of the future operators are not used, iff they do not contain any past operator. Removing the over head would improve the performance, if the check of the presents of a past operator is faster and there is no past operator present. This optimisation is only used at the initial answer formula. In the next answer formula, the results are used by the update formula.

Another improvement is the reduction of subformula duplicates. It reduces the computation, if a query contains equal subformulas. This improvement changes the parser, which has to recognise duplicates. Furthermore, the recursion of the query has to recognise duplicates, because otherwise there will be no improvement.

A very important part of future work is the import of sequences. The sequences can be finite or infinite as dynamic stream. If a sequence is finite, the algorithm starts with the initial answer formula. If the sequence is infinite as stream and one get an interval of A-Boxes, the algorithm can treat them as finite sequences or try to re-use the results of the previous calculated answer formulas. This is only possible, if there is no gab between two intervals and overlapping intervals are recognised.

6. Conclusion

All in all, i introduced an implementation of an algorithm of answering DL-Lite queries on A-Box sequences according to [2]. I explained the theoretical part and implemented the algorithm. Afterwards i extended the implementation with the Box and Diamond operator, the exists operator and the concept of concrete domains.

In the future work chapter, i represented possible improvements. The main part is the import of sequences. My current implementation uses a repository, which has already the sequence in memory. In reality one get sequences dynamically, which have no clear labelling for the A-Boxes. Two sequences are not necessary next to each other. They can overlap or have a gap between. Therefore a solution for processing sequences as stream has to be created.

This query answering over streams has often the same query to answer. A pre-compiled query can improve the performance by an one time optimisation. This pre-compiled version of the query could cancel recursions in the initial answer formula as presented in the future work chapter. It could identify over heads and reduce the amount of control flow elements in the implementation.

With the help of a fixed sequence of A-Boxes, i run manual tests to check the correctness of the implementation. As mentioned in chapter 4, i have a correct working algorithm, as described in [2].

Bibliography

[1] Stefan Borgwardt, Marcel Lippmann, and Veronika Thost. Temporal query answering in *DL-Lite*. In Thomas Eiter, Birte Glimm, Yevgeny Kazakov, and Markus Krtzsch, editors, *Description Logics*, volume 1014 of *CEUR Workshop Proceedings*, pages 80–92. CEUR-WS.org, 2013.

[2] Stefan Borgwardt, Marcel Lippmann, and Veronika Thost. Temporal query answering w.r.t. *DL-Lite*-ontologies. LTCS-Report 13-05, Chair of Automata Theory, TU Dresden, Dresden, Germany, 2013. Revised version. See http://lat.inf.tu-dresden.de/research/reports.html.

[3] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, and Riccardo Rosati. Ontologies and databases: *The DL-Lite Approach*. In Sergio Tessaris, Enrico Franconi, Thomas Eiter, Claudio Gutierrez, Siegfried Handschuh, Marie-Christine Rousset, and Renate A. Schmidt, editors, *Reasoning Web*, volume 5689 of *Lecture Notes in Computer Science*, pages 255–356. Springer, 2009.

[4] Terence Parr and Kathleen Fisher. Ll(*): *The Foundation of the ANTLR Parser Generator*. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 425–436, New York, NY, USA, 2011. ACM.

A. The Implementation

This chapter describes the implementation of the algorithm. Figure A.1 shows the grammar of the parser. The parser is implemented with the help of antlr [4].

The first algorithms implement the answer formula operations. An answer formula is represented by the Java Class *assignment*. The set of answer formulas is represented by the Java Class *assignments*. The union of two sets of answer formulas is described in algorithm 2. It uses a helper function (Algorithm 3) to calculate the new boundaries in terms of concrete domains and only accept elements, that are not outside this boundaries. The list $assignments.min$ represents the lower bounds and $assignments.max$ the upper bounds, where x represents the variable, op the operator, and v the value.

The Algorithm 4 describes the intersection of two sets of answer formulas. It has a helper function (Algorithm 5) to calculate the new boundaries, too. For every pair of answer formulas, it calls a helper function (Algorithm 6) to compute the intersection. An answer formula is $\top$, if it represents all mappings. This $\top$ is denoted by *top*. The intersection of two answer formulas returns one of the answer formulas, if the other answer formula is $\top$ and in the boundaries. If none of the answer formulas is top, the intersection is calculated. A mapping is in the result answer formula, if it is in the boundaries and the other set has no mapping of this variable or it has the same value.

The Algorithm 7 represents a helper function to remove mappings of a specific set of variables. This function is used by the exists operator implementation in the initial answer formula and the next answer formula.

The initial answer formula (Algorithm 8, 9) is a recursive function over the elements of the query. Depending on the type of query element, it calculates the initial answer formula set. A query element is denoted by *query*. $query(n)$ denotes the n-th subformula, starting by 0. $variable(query, i)$ denotes a placeholder variable of the subformula *query* at point in time i. The helper function (Algorithm 11) extracts the concrete domain of the query and represents it as a set of answer formulas. The second helper function (Algorithm 10) queries the assertions against the database. The algorithm saves the answer formulas of two points in time. The current point

```
grammar Query;

tcq: tcq_until | tcq_since | '(' tcq ')' | tcq 'AND' tcq
    | tcq 'OR' tcq | tcq_sn | tcq_sp| tcq_wn | tcq_wp
    | tcq_box | tcq_dia | tcq_box_past | tcq_dia_past | cq;

tcq_until:      '(' tcq ')' 'U' '(' tcq ')';
tcq_since:      '(' tcq ')' 'S' '(' tcq ')';
tcq_sn:         'SN(' tcq ')';
tcq_sp:         'SP(' tcq ')';
tcq_wn:         'WN(' tcq ')';
tcq_wp:         'WP(' tcq ')';
tcq_box:        'BOX(' tcq ')';
tcq_dia:        'DIA(' tcq ')';
tcq_box_past:   'BOXP(' tcq ')';
tcq_dia_past:   'DIAP(' tcq ')';

cq:   cq 'AND' cq | cq 'OR' cq | '(' cq ')'
  | cq_single | cq_double | cq_ex | cq_alg_one | cq_alg_two;

cq_ex:          'EX[' (VAR (',' VAR)*) '](' cq ')';
cq_single:      URL '(' (URL|VAR) ')';
cq_double:      URL '(' (URL|VAR) ',' (VAR|LIT|URL) ')';

cq_alg_one:             (VAR|LIT) ('<'|'<='|'='|'>'|'>=') (LIT);
cq_alg_two:             (LIT) ('<'|'<='|'='|'>'|'>=') (VAR|LIT);

INT : ('0'..'9')+;
URL : 'http' (~[\(\),])+;
VAR : ('a'..'z')+;
LIT : ('"'~[\(\),' '\[\]]+);

WHITESPACE : ('\t' | ' ' | '\r' | '\n' | '\u000c' )+ -> skip ;
```

Figure A.1.: Antlr grammar for the queries

Algorithm 2 *assignments.union*

Input: *assignments A*, *assignments B*
Output: *assignments C*

```
C ← Assignments.extrema.union(A, B);
for a ∈ A do
  if ∀x_a ∈ a.Vars : ∄B.min(x_a, op_b, v_b) ∨ a(x_a) op_b v_b then
    if ∀x_a ∈ a.Vars : ∄B.max(x_a, op_b, v_b) ∨ a(x_a) op_b v_b then
      C ← a;
    end if
  end if
end for
for b ∈ B do
  if ∀x_b ∈ b.Vars : ∄A.min(x_b, op_a, v_a) ∨ a(x_b) op_a v_a then
    if ∀x_b ∈ b.Vars : ∄A.max(x_b, op_a, v_a) ∨ a(x_b) op_a v_a then
      C ← b;
    end if
  end if
end for
return C;
```

in time and the previous one. The storage to store the current point in time is $STORAGE_{Current}$. $STORAGE_{Previous}$ denotes the storage of the previous point in time.

The next answer formula (Algorithm 12, 13, 14) has the same structure as the initial answer formula. The difference between both algorithm is the usage of the previous answer formulas. The past operators use the previous calculated answer formulas of $STORAGE_{Previous}$.

The update formula (Algorithm 15) is a recursive function. It traversal the query structure of subformulas in post-order. This ensures, that every subformula has an updated answer formula. With the help of Algorithm 16, the placeholder variables of the previous point in time are replaced with answer formulas of the current point in time. Furthermore, intersections and unions are calculated, if no placeholder variable prevent it.

The evaluation function (Algorithm 17) uses the set of answer formulas of the whole query and replace all occurrences of placeholder variables. Only placeholder variables of the current point in time are present. It calculate all intersections and unions and results into one set of answer formulas.

The consume function (Algorithm 18) is called by the main function. Its task is to consume an A-Box of the sequence. If it is the first A-Box in the sequence,

Algorithm 3 *assignments.extrema.union*

Input: *assignments* A, *assignments* B
Output: *assignments* C

for $(x_a, op_a, v_a) \in A.min(x, op, v)$ **do**
  if $\exists B.min(x_a, op_b, v_b)$ **then**
    if $v_a\ op_a\ v_b$ **then**
      $C.min \leftarrow (x_b, op_b, v_b)$;
    else
      $C.min \leftarrow (x_a, op_a, v_a)$;
    end if
  else
    $C.min \leftarrow (x_a, op_a, v_a)$;
  end if
end for
for $(x_b, op_b, v_b) \in B.min(x, op, v) \wedge \neg\exists A.min(x_b, op, v)$ **do**
  $C.min \leftarrow (x_b, op_b, v_b)$;
end for
for $(x_a, op_a, v_a) \in A.max(x, op, v)$ **do**
  if $\exists B.max(x_a, op_b, v_b)$ **then**
    if $v_a\ op_a\ v_b$ **then**
      $C.max \leftarrow (x_a, op_a, v_a)$;
    else
      $C.max \leftarrow (x_b, op_b, v_b)$;
    end if
  else
    $C.max \leftarrow (x_a, op_a, v_a)$;
  end if
end for
for $(x_b, op_b, v_b) \in B.max(x, op, v) \wedge \neg\exists A.max(x_b, op, v)$ **do**
  $C.max \leftarrow (x_b, op_b, v_b)$;
end for
return C;

Algorithm 4 *assignments.intersection*

Input: *assignments* A, *assignments* B
Output: *assignments* C

```
C ← Assignments.extrema.intersection(A, B);
for a ∈ A do
  for b ∈ B do
    if assignment.intersection(a, b, C.min, C.max) ≠ ∅ then
      C ← assignment.intersection(a, b)
    end if
  end for
end for
return C;
```

the initial answer formula is called. Otherwise i move the current storage to the previous one and call the next answer formula followed by the update formula. To distinguish the placeholder variables, i denotes the current position in the sequence.

The main function (Algorithm 19) calls for every A-Box in the sequence the consume function. Afterwards it calls the evaluation function to eliminate all placeholder variables of the set of answer formulas of the last point in time.

Algorithm 5 *assignments.extrema.intersection*

Input: *assignments* A, *assignments* B
Output: *assignments* C

```
for (x_a, op_a, v_a) ∈ A.min(x, op, v) do
  if ∃B.min(x_a, op_b, v_b) then
    if v_a op_a v_b then
      C.min ← (x_a, op_a, v_a);
    else
      C.min ← (x_b, op_b, v_b);
    end if
  else
    C.min ← (x_a, op_a, v_a);
  end if
end for
for (x_b, op_b, v_b) ∈ B.min(x, op, v) ∧ ¬∃A.min(x_b, op, v) do
  C.min ← (x_b, op_b, v_b);
end for
for (x_a, op_a, v_a) ∈ A.max(x, op, v) do
  if ∃B.max(x_a, op_b, v_b) then
    if v_a op_a v_b then
      C.max ← (x_b, op_b, v_b);
    else
      C.max ← (x_a, op_a, v_a);
    end if
  else
    C.max ← (x_a, op_a, v_a);
  end if
end for
for (x_b, op_b, v_b) ∈ B.max(x, op, v) ∧ ¬∃A.max(x_b, op, v) do
  C.max ← (x_b, op_b, v_b);
end for
return C;
```

Algorithm 6 *assignment.intersection*

Input: *assignment* A, *assignment* B, max, min
Output: *assignment* C

```
if A.top then
  if ∀x_b ∈ B.Vars : ∄A.min(x_b, op_a, v_a) ∨ a(x_b) op_a v_a then
    if ∀x_b ∈ B.Vars : ∄A.max(x_b, op_a, v_a) ∨ a(x_b) op_a v_a then
      return B;
    end if
  end if
  return ∅;
end if
if B.top then
  if ∀x_a ∈ A.Vars : ∄min(x_a, op_b, v_b) ∨ a(x_a) op_b v_b then
    if ∀x_a ∈ A.Vars : ∄max(x_a, op_b, v_b) ∨ a(x_a) op_b v_b then
      return A;
    end if
  end if
  return ∅;
end if
for a ∈ A do
  if a.var ∈ B.Var then
    if a(a.var) = b(a.var) then
      C ← a;
    else
      return ∅
    end if
  else
    if ∄min(a.var, op, v) ∨ a(a.var) op v then
      if ∄max(a.var, op, v) ∨ a(a.var) op v then
        C ← a;
      else
        return ∅;
      end if
    else
      return ∅;
    end if
  end if
end for
for b ∈ B do
  if b.var ∉ A.Var then
    if ∄min(b.var, op, v) ∨ b(b.var) op v then
      if ∄max(b.var, op, v) ∨ b(b.var) op v then
        C ← b;
      else
        return ∅;
      end if
    else
      return ∅;
    end if
  end if
end for
return C;
```

Algorithm 7 $assignments.removeVars$

Input: $assignments\ A, Vars$
Output: $assignments\ C$

```
for assignment_a ∈ A do
  assignment_new = ∅;
  for a ∈ assignment_a do
    if a.var ∉ Vars then
      assignment_new ← a;
    end if
    if assignment_new ≠ ∅ then
      C ← assignment_new;
    end if
  end for
end for
if C = ∅ ∧ A ≠ ∅ then
  assignment.top = true;
  C ← assignment;
end if
return C;
```

Algorithm 8 initialAnswerFormula (Part 1)

Input: Query *query*

```
if query ⊆ SingleAssertion ∨ query ⊆ DoubleAssertion then
    helperFunctionAssertions(query);
end if
if query ⊆ SN ∨ query ⊆ WN ∨ query ⊆ BOXP ∨ query ⊆ DIAP then
    initialAnswerFormula(query(0));
    STORAGE_Current ← (query, variable(query, 0));
    return
end if
if query ⊆ SP then
    initialAnswerFormula(query(0));
    STORAGE_Current ← (query, assignments);
    return
end if
if query ⊆ WP then
    initialAnswerFormula(query(0));
    assignment.top = true;
    assignments ← assignment;
    STORAGE_Current ← (query, assignments);
    return
end if
if query ⊆ Since then
    initialAnswerFormula(query(0));
    initialAnswerFormula(query(1));
    STORAGE_Current ← (query, STORAGE_Current(query(1));
    return
end if
if query ⊆ Until then
    initialAnswerFormula(query(0));
    initialAnswerFormula(query(1));
    assignments = intersection(STORAGE_Current(query(0)), variable(query, 0));
    assignments = union(STORAGE_Current(query(1)), assignments);
    STORAGE_Current ← (query, assignments);
    return
end if
if query ⊆ BOX then
    initialAnswerFormula(query(0));
    assignments = intersection(STORAGE_Current(query(0)), variable(query, 0));
    STORAGE_Current ← (query, assignments);
    return
end if
```

Algorithm 9 initialAnswerFormula (Part 2)

if $query \subseteq DIA$ **then**
  $initialAnswerFormula(query(0))$;
  $assignments = union(STORAGE_{Current}(query(0)), variable(query, 0))$;
  $STORAGE_{Current} \leftarrow (query, assignments)$;
  return
end if
if $query \subseteq EXISTS$ **then**
  $initialAnswerFormula(query(0))$;
  $assignments = STORAGE_{Current}(query(0)).removeVars(EXISTS.Vars)$;
  $STORAGE_{Current} \leftarrow (query, assignments)$;
  return
end if
if $query \subseteq Conjunction$ **then**
  $initialAnswerFormula(query(0))$;
  $initialAnswerFormula(query(1))$;
  $assignments_a = STORAGE_{Current}(query(0))$;
  $assignments_b = STORAGE_{Current}(query(1))$;
  $assignments = intersection(assignments_a, assignments_b)$;
  if $assignments(0) \subseteq Assignments \wedge assignments(1) \subseteq Assignments$ **then**
    $assignments = Assignments.intersection(assignments(0), assignments(1))$;
  end if
  $STORAGE_{Current} \leftarrow (query, assignments)$;
  return
end if
if $query \subseteq Disjunction$ **then**
  $initialAnswerFormula(query(0))$;
  $initialAnswerFormula(query(1))$;
  $assignments_a = STORAGE_{Current}(query(0))$;
  $assignments_b = STORAGE_{Current}(query(1))$;
  $assignments = union(assignments_a, assignments_b)$;
  if $assignments(0) \subseteq Assignments \wedge assignments(1) \subseteq Assignments$ **then**
    $assignments = Assignments.union(assignments(0), assignments(1))$;
  end if
  $STORAGE_{Current} \leftarrow (query, assignments)$;
  return
end if
if $query \subseteq EQUATION$ **then**
  $STORAGE_{Current} \leftarrow (query, getConreteDomain(query(0)))$;
  return
end if

Algorithm 10 helperFunctionAssertions

Input: Query *query*

```
if query ⊆ SingleAssertion then
  if Assertion A(x), where x is a variable then
    while Database has results do
      assignment = {x → value};
      assignments ← assignment;
    end while
  else
    if Assertion exists then
      assignment.top = true;
      assignments ← assignment;
    end if
  end if
  STORAGE_Current ← (query, assignments);
  return
end if
if query ⊆ DoubleAssertion then
  if Assertion A(x, y), where x and y are variables then
    while Database has results do
      assignment ← {x → value_x, y → value_y};
      assignments ← assignment;
    end while
  else
    if Assertion A(x, v) or A(v, x), where x is a variable and v a value then
      while Database has results do
        assignment ← {x → value_x};
        assignments ← assignment;
      end while
    else
      if Assertion exists then
        assignment.top = true;
        assignments ← assignment;
      end if
    end if
  end if
  STORAGE_Current ← (query, assignments);
end if
```

Algorithm 11 getConreteDomain

Input: EQUATION *equation*
Output: *assignments*

```
if equation(0) ⊆ Value then
  if equation(1) ⊆ Value then
    if equation(0) equation.operator equation(1) then
      assignment.top = true;
      assignments ← assignment;
    end if
  else
    if equation.operator ∈ {<, ≤} then
      assignments.min ← (equation(1), equation.operator, equation(0));
    end if
    if equation.operator ∈ {>, ≥} then
      assignments.max ← (equation(1), equation.operator, equation(0));
    end if
    if equation.operator ∈ {=} then
      assignments.max ← (equation(1), ≥, equation(0));
      assignments.min ← (equation(1), ≤, equation(0));
    end if
  end if
else
  if equation.operator ∈ {<, ≤} then
    assignments.max ← (equation(1), equation.operator, equation(0));
  end if
  if equation.operator ∈ {>, ≥} then
    assignments.min ← (equation(1), equation.operator, equation(0));
  end if
  if equation.operator ∈ {=} then
    assignments.max ← (equation(1), ≥, equation(0));
    assignments.min ← (equation(1), ≤, equation(0));
  end if
end if
return assignments;
```

Algorithm 12 nextAnswerFormula (Part 1)

Input: Query $query$ and Point in Time i

```
if query ⊆ SingleAssertion ∨ query ⊆ DoubleAssertion then
  helperFunctionAssertions(query);
end if
if query ⊆ SN ∨ query ⊆ WN then
  nextAnswerFormula(query(0), i);
  STORAGE_Current ← (query, variable(query, i));
  return
end if
if query ⊆ SP ∨ query ⊆ WP then
  nextAnswerFormula(query(0), i);
  STORAGE_Current ← (query, STORAGE_Previous(query(0)));
  return
end if
if query ⊆ Since then
  nextAnswerFormula(query(0), i);
  nextAnswerFormula(query(1), i);
  assignments = STORAGE_Current(query(0));
  assignments = intersection(assignments, STORAGE_Previous(query(0)));
  assignments = union(STORAGE_Current(query(1)), assignments);
  STORAGE_Current ← (query, assignments);
  return
end if
if query ⊆ Until then
  nextAnswerFormula(query(0), i);
  nextAnswerFormula(query(1), i);
  assignments = STORAGE_Current(query(0));
  assignments = intersection(assignments, variable(query, i));
  assignments = union(STORAGE_Current(query(1)), assignments);
  STORAGE_Current ← (query, assignments);
  return
end if
if query ⊆ BOXP then
  nextAnswerFormula(query(0), i);
  assignments_a = STORAGE_Previous(query);
  assignments_b = STORAGE_Current(query(0));
  assignments = intersection(assignments_a, assignments_b);
  if assignments(0) ⊆ Assignments ∧ assignments(1) ⊆ Assignments then
    assignments = Assignments.intersection(assignments(0), assignments(1));
  end if
  STORAGE_Current ← (query, assignments);
  return
end if
```

Algorithm 13 nextAnswerFormula (Part 2)

if $query \subseteq DIAP$ **then**
  $nextAnswerFormula(query(0), i)$;
  $assignments_a = STORAGE_{Previous}(query)$;
  $assignments_b = STORAGE_{Current}(query(0))$;
  $assignments = union(assignments_a, assignments_b)$;
  if $assignments(0) \subseteq Assignments \wedge assignments(1) \subseteq Assignments$ **then**
    $assignments = Assignments.union(assignments(0), assignments(1))$;
  end if
  $STORAGE_{Current} \leftarrow (query, assignments)$;
  return
end if
if $query \subseteq BOX$ **then**
  $nextAnswerFormula(query(0), i)$;
  $assignments = intersection(STORAGE_{Current}(query(0)), variable(query, i))$;
  $STORAGE_{Current} \leftarrow (query, assignments)$;
  return
end if
if $query \subseteq DIA$ **then**
  $nextAnswerFormula(query(0), i)$;
  $assignments = union(STORAGE_{Current}(query(0)), variable(query, i))$;
  $STORAGE_{Current} \leftarrow (query, assignments)$;
  return
end if
if $query \subseteq EXISTS$ **then**
  $nextAnswerFormula(query(0), i)$;
  $assignments = STORAGE_{Current}(query(0)).removeVars(EXISTS.Vars)$;
  $STORAGE_{Current} \leftarrow (query, assignments)$;
  return
end if
if $query \subseteq Conjunction$ **then**
  $nextAnswerFormula(query(0), i)$;
  $nextAnswerFormula(query(1), i)$;
  $assignments_a = STORAGE_{Current}(query(0))$;
  $assignments_b = STORAGE_{Current}(query(1))$;
  $assignments = intersection(assignments_a, assignments_b)$;
  if $assignments(0) \subseteq Assignments \wedge assignments(1) \subseteq Assignments$ **then**
    $assignments = Assignments.intersection(assignments(0), assignments(1))$;
  end if
  $STORAGE_{Current} \leftarrow (query, assignments)$;
  return
end if

Algorithm 14 nextAnswerFormula (Part 3)

if $query \subseteq Disjunction$ **then**
  $nextAnswerFormula(query(0), i)$;
  $nextAnswerFormula(query(1), i)$;
  $assignments_a = STORAGE_{Current}(query(0))$;
  $assignments_b = STORAGE_{Current}(query(1))$;
  $assignments = union(assignments_a, assignments_b)$;
  if $assignments(0) \subseteq Assignments \wedge assignments(1) \subseteq Assignments$ **then**
    $assignments = Assignments.union(assignments(0), assignments(1))$;
  end if
  $STORAGE_{Current} \leftarrow (query, assignments)$;
  return
end if
if $query \subseteq EQUATION$ **then**
  $STORAGE_{Current} \leftarrow (query, getConreteDomain(query(0)))$;
  return
end if

Algorithm 15 updateFormula

Input: Query *query* and Point in Time i

```
if query ⊆ SingleAssertion ∨ query ⊆ DoubleAssertion then
  return
end if
if query ⊆ SN ∨ query ⊆ WN then
  updateFormula(query(0), i);
end if
if query ⊆ SP ∨ query ⊆ WP then
  updateFormula(query(0), i);
end if
if query ⊆ Since then
  updateFormula(query(0), i);
  updateFormula(query(1), i);
end if
if query ⊆ Until then
  updateFormula(query(0), i);
  updateFormula(query(1), i);
end if
if query ⊆ Conjunction ∨ query ⊆ Disjunction then
  updateFormula(query(0), i);
  updateFormula(query(1), i);
end if
if query ⊆ BOX ∨ query ⊆ DIA then
  updateFormula(query(0), i);
end if
if query ⊆ EXISTS then
  updateFormula(query(0), i);
end if
assignments = updateAnswerFormulas(STORAGE_Current(query));
STORAGE_Current ← (query, assignments);
```

Algorithm 16 updateAnswerFormulas

Input: *assignments*
Output: *assignments*

```
if assignments ⊆ Intersection then
  assignments_1 = updateAnswerFormulas(assignments(0));
  assignments_2 = updateAnswerFormulas(assignments(1));
  assignments = intersection(assignments_1, assignments_2);
  if assignments(0) ⊆ Assignments ∧ assignments(1) ⊆ Assignments then
    assignments = Assignments.intersection(assignments(0), assignments(1));
  end if
  return assignments;
end if
if assignments ⊆ Union then
  assignments_1 = updateAnswerFormulas(assignments(0));
  assignments_2 = updateAnswerFormulas(assignments(1));
  assignments = union(assignments_1, assignments_2);
  if assignments(0) ⊆ Assignments ∧ assignments(1) ⊆ Assignments then
    assignments = Assignments.union(assignments(0), assignments(1));
  end if
  return assignments;
end if
if assignments ⊆ Variable then
  if assignments.query ⊆ SN ∨ assignments.query ⊆ WN then
    return STORAGE_Current(assignments.query(0));
  end if
  if assignments.query ⊆ BOX ∨ assignments.query ⊆ DIA then
    return STORAGE_Current(assignments.query(0));
  end if
  if assignments.query ⊆ Until then
    return STORAGE_Current(assignments.query);
  end if
end if
return assignments;
```

Algorithm 17 evaluationFormula

Input: $assignments$
Output: $assignments$

```
if assignments ⊆ Intersection then
  assignments_1 = evaluationFormula(assignments(0));
  assignments_2 = evaluationFormula(assignments(1));
  assignments = intersection(assignments_1, assignments_2);
  if assignments(0) ⊆ Assignments ∧ assignments(1) ⊆ Assignments then
    assignments = Assignments.intersection(assignments(0), assignments(1));
  end if
  return assignments;
end if
if assignments ⊆ Union then
  assignments_1 = evaluationFormula(assignments(0));
  assignments_2 = evaluationFormula(assignments(1));
  assignments = union(assignments_1, assignments_2);
  if assignments(0) ⊆ Assignments ∧ assignments(1) ⊆ Assignments then
    assignments = Assignments.union(assignments(0), assignments(1));
  end if
  return assignments;
end if
if assignments ⊆ Variable then
  if assignments.query ⊆ SN ∨ assignments.query ⊆ Until then
    return assignments;
  end if
  if assignments.query ⊆ DIA then
    return assignments;
  end if
  if assignments.query ⊆ WN ∨ assignments.query ⊆ BOX then
    assignment.top = true;
    assinments ← assignment;
    return assignments;
  end if
end if
return assignments;
```

Algorithm 18 Consume Function

Input: $A - Box$
  if $i = 0$ **then**
    $STORAGE_{Current} = \emptyset$;
    $initialAnswerFormulas(query)$;
  else
    $STORAGE_{Previous} = STORAGE_{Current}$;
    $STORAGE_{Current} = \emptyset$;
    $nextAnswerFormulas(query, i)$;
    $updateFormulas(query, i)$;
  end if
  $i = i + 1$;

Algorithm 19 Main Function

Input: *Sequence* of $A - Box$, Query *query*
Output: *assignments*
  $algo.query = query$;
  for $abox \in Sequence$ **do**
    $algo.consumeFormula(abox)$;
  end for
  return $algo.evaluationFormula(STORAGE_{Current}(algo.query))$;