EPiC Series in Computing

Volume 65, 2019, Pages 15–27

GCAI 2019. Proceedings of the 5th Global Conference on Artificial Intelligence

# Practical Query Rewriting for DL-Lite with Numerical Predicates[*]

Christian Alrabbaa, Patrick Koopmann, and Anni-Yasmin Turhan

Theoretical Computer Science, Technische Universität Dresden, Germany
`firstname.lastname@tu-dresden.de`

**Abstract**

We present a method for answering ontology-mediated queries for DL-Lite extended with a concrete domain, where we allow concrete domain predicates to be used in the query as well. Our method is based on query rewriting, a well-known technique for ontology-based query answering (OBQA), where the knowledge provided by the ontology is compiled into the query so that the rewritten query can be evaluated directly over a database. This technique reduces the problem of query answering w.r.t. an ontology to query evaluation over a database instance. Specifically, we consider members of the DL-Lite family extended with unary and binary concrete domain predicates over the real numbers. While approaches for query rewriting DL-Lite with these concrete domain have been investigated theoretically, these approaches use a combined approach in which also the data is processed, and require the concrete domain values occurring in the data to be known in advance, which makes the procedure data-dependent. In contrast, we show how rewritings can be computed in a data-independent fashion.

## 1 Introduction

Formal ontologies are useful to augment application data in order to be able to extract more consequences from the data by use of background knowledge than from a query over the plain data alone. In recent years, *ontology-based query answering* (OBQA) by means of Description Logics (DLs) has become a prominent example of this setting. In many practical applications such as medical or stream-reasoning applications, where data is produced by sensors, the data need not always be symbolic, but can be numerical. Concepts from such applications can be characterized by relating their instances to numerical values. For example, patients with high blood pressure can be modelled as patients with a value for blood pressure over 180. Such statements can be expressed in an ontology by the use of concrete domains [5].

In OBQA applications, the expressiveness of the underlying DL can lead to high complexity of query answering, which limits a fast execution of ontology-based queries [14, 12, 10]. This has lead to the development of the DL-Lite family of DLs that are designed such that their expressiveness admits to perform query answering by the well-known rewriting approach for

[*]Supported by the DFG in the CRC 912 (HAEC), in the TRR 248 (CPEC), and in the GRK 1763 (QuantLA).

answering conjunctive queries [7, 2]. In the classical rewriting approach, the query is rewritten such that the resulting query incorporates the relevant knowledge from the ontology. Then, the rewritten query is answered over the plain (or possibly enriched) data by a database engine directly. DLs that admit this approach are called *first-order rewritable.* The rewriting approach has strong benefits. Since answering conjunctive queries has a complexity of $\mathsf{AC}^0$ measured in the size of the data, rewritability means that query answering is of the same complexity. Furthermore, the rewriting is *data-independent*, and thus only has to be performed once, afterwards the rewritten query can be executed on different databases without further adaptations or further reasoning steps. This is especially useful for querying big data or frequently changing data. Another approach to solve OBQA by means of standard database query answering is the *combined rewriting approach*, in which the data is enriched based on the ontology before the rewritten query is executed [11, 9, 8].

Combinations of DL-Lite and concrete domains have been investigated in regard of query answering early on [13, 15, 3]. In these combinations either the concrete domain predicates are only unary [13, 15, 3] or the query language does not admit the use of concrete domain predicates [13]. Both restrictions are severe limitations on the expressiveness.

Recently, Baader et al. identified in [4] a criterion for concrete domains with $n$-ary predicates that admits *combined rewritability* when used in combination with DL-Lite. This so-called *cr-admissibility* consists of several properties that the concrete domain must fulfill. Among others, the concrete domain must be convex and admit polynomial reasoning, it must contain equality in its set of predicates, and it must be functional, i.e. for any predicate (of non-zero arity) applied to a tuple of variables, where one of these variables has a fixed value, there is at most one solution. See [4] for a discussion of all the properties required by cr-admissibility.

Two concrete domains that are identified as cr-admissible in [4], are those over the rational numbers with predicates including equality and one comparison $\sim_d \in \{<_d, >_d\}$ to arbitrary values $d$ and a predicate to state a distance $+_d(v, w)$ from one value to another. The presence of both comparison operators or a binary predicate $<_d$ could be desirable, but it would destroy convexity of the concrete domain. This concrete has infinitely many different unary and also binary predicates. The latter gives more expressiveness as the concrete domains considered in earlier approaches and admits to express the example from above:

$$\mathsf{HighBloodPressurePatient} \sqsubseteq \mathsf{Patient} \sqcap \exists \mathsf{hasBloodPressure}. >_{180} .$$

Furthermore it can express that high risk patients are patients whose systolic blood pressure is above their diastolic blood pressure by 90 (mmHg):

$$\mathsf{HighRiskPatient} \sqsubseteq \mathsf{Patient} \sqcap \exists \mathsf{hasDiastolicBloodPressure}, \mathsf{hasSystolicBloodPressure}.+_{90} .$$

Despite being a polynomial method for evaluating queries, the technique proposed in [4] is a combined rewriting approach, which means that the data has to be processed before the rewritten query can be executed. Furthermore, the query rewriting procedure requires full knowledge of the concrete domain values that occur in the data. This limits the practical applicability of the technique for large or frequently changing data sets—one of the main advantages of DLs admitting full first-order rewritability. To solve this issue, we present a *data-independent* rewriting procedure for DL-Lite extended with the aforementioned cr-admissible concrete domains.

Our rewriting procedure proceeds in two phases. First, it saturates the terminological part of the ontology, and second, it rewrites the input query based on the saturated ontology. When answering conjunctive queries by a data-independent rewriting approach, the rewritten query must cater for all possibilities how concept membership or satisfaction of a concrete domain predicate can be derived based on the information in the ontology. For instance, if a concept implies a positive distance between a pair of values, and the ontology also states a bound for the first value, then a bound on the second value can be inferred. Such information does only depend on the ontology and not on the data. In the first phase, our rewriting approach makes such information explicit and adds it to the TBox in the form of new axioms. This TBox saturation can be done even independently of the query. In the second phase, our algorithm computes a rewriting of a given query in regard to the saturated TBox. In contrast to the classical rewriting for DL-Lite without concrete domains, here the challenge is that our rewriting procedure needs to cope with a potentially infinite set of predicates.

This paper is structured as follows. The next section introduces the concrete domains $\mathbb{R}_>$ and $\mathbb{R}_<$, the resulting logic DL-Lite$_\sqcap(\mathbb{R}_\sim)$ and answering of conjunctive queries. Section 3 gives an overview of the rewriting method and Section 4, describes the TBox saturation and its properties. Section 5 introduces the algorithm to compute the query rewriting and we show that it is complete and terminating. Finally, we provide our conclusion in Section 6. Proof details of our results can be found in the extended version of the paper [1].

## 2 Query Answering in DL-Lite$_\sqcap(\mathbb{R}_>)$ and DL-Lite$_\sqcap(\mathbb{R}_<)$

We recall syntax and semantics of DL-Lite$_\sqcap(\mathcal{D})$ and the main task conjunctive query answering.

### 2.1 The Concrete Domains $\mathbb{R}_>$ and $\mathbb{R}_<$

We define two concrete domains over the real numbers, $\mathbb{R}_>$ and $\mathbb{R}_<$, that are used in our DL. In general, a concrete domain [5] is a tuple $\mathcal{D} = \langle \Delta^\mathcal{D}, \mathcal{P}^\mathcal{D}, \mathsf{ar}^\mathcal{D}, \cdot^\mathcal{D}\rangle$ of a set $\Delta^\mathcal{D}$ of *concrete domain elements*, a set of $\mathcal{P}^\mathcal{D}$ of *concrete domain predicates*, where to each $\Pi \in \mathcal{P}$ an arity $\mathsf{ar}(\Pi) \in \mathbb{N}$ is associated, and an *interpretation function* $\cdot^\mathcal{D}$ which assigns to each $\Pi \in \mathcal{P}$ with $\mathsf{ar}(\Pi) = n$ a set $\Pi^\mathcal{D} \subseteq (\Delta^\mathcal{D})^n$. We focus on two concrete domains, $\mathbb{R}_>$ and $\mathbb{R}_<$, defined by $\mathbb{R}_\sim = \langle \mathbb{R}, \mathcal{P}^{\mathbb{R}_\sim}, \mathsf{ar}^{\mathbb{R}_\sim}, \cdot^{\mathbb{R}_\sim}\rangle$, for one comparison predicate $\sim \in \{<, >\}$ per concrete domain, with the set $\mathcal{P}^{\mathbb{R}_\sim}$ of predicates defined as $\mathcal{P}^{\mathbb{R}_\sim} = \{\top^1_{\mathbb{R}_\sim}, \top^2_{\mathbb{R}_\sim}, \bot^1_{\mathbb{R}_\sim}, \bot^2_{\mathbb{R}_\sim}\} \cup \{=_d, \sim_d, +_d \mid d \in \mathbb{R}\}$, arities $\mathsf{ar}^{\mathbb{R}_\sim}(=_d) = \mathsf{ar}^{\mathbb{R}_\sim}(\sim_d) = \mathsf{ar}^{\mathbb{R}_\sim}(\top^1_{\mathbb{R}_\sim}) = \mathsf{ar}^{\mathbb{R}_\sim}(\bot^1_{\mathbb{R}_\sim}) = 1$ and $\mathsf{ar}^{\mathbb{R}_\sim}(+_d) = \mathsf{ar}^{\mathbb{R}_\sim}(\top^2_{\mathbb{R}_\sim}) = \mathsf{ar}^{\mathbb{R}_\sim}(\bot^2_{\mathbb{R}_\sim}) = 2$, and an interpretation function defined as

$$
\begin{array}{lll}
(\top^1_{\mathbb{R}_\sim})^{\mathbb{R}_\sim} = \mathbb{R} & (\bot^1_{\mathbb{R}_\sim})^{\mathbb{R}_\sim} = (\bot^2_{\mathbb{R}_\sim})^{\mathbb{R}_\sim} = \emptyset & (\sim_d)^{\mathbb{R}_\sim} = \{d' \mid d' \in \mathbb{R}, d' \sim d\} \\
(\top^2_{\mathbb{R}_\sim})^{\mathbb{R}_\sim} = \mathbb{R} \times \mathbb{R} & (=_d)^{\mathbb{R}_\sim} = \{d\} & (+_d)^{\mathbb{R}_\sim} = \{\langle d_1, d_2\rangle \mid d_1, d_2 \in \mathbb{R}, d_1 + d = d_2\}.
\end{array}
$$

Given two predicates $\Pi_a$ and $\Pi_b$ of the same arity, we write $\Pi_a \models \Pi_b$ iff $(\Pi_a)^{\mathbb{R}_\sim} \subseteq (\Pi_b)^{\mathbb{R}_\sim}$.

### 2.2 The Description Logics DL-Lite$_\sqcap(\mathbb{R}_\sim)$

We recall DL-Lite$_\sqcap(\mathbb{R}_\sim)$ with the two concrete domains just introduced. Let $\mathsf{N_C}$, $\mathsf{N_R}$, $\mathsf{N_A}$ and $\mathsf{N_I}$ be pairwise disjoint, countably infinite sets of respectively *concept names*, *role names*, *attribute names* and *individual names*. A *role* $R$ is an expression of the form $r$ or $r^-$, where $r \in \mathsf{N_R}$. DL-Lite$_\sqcap(\mathcal{D})$ *concepts* $C$, $D$ and *axioms* $\mathfrak{a}$ are defined according to the following syntax rule, where $A \in \mathsf{N_C}$, $R$, $S$ are roles, $U_1, \dots, U_n \in \mathsf{N_A}$, and $\Pi_n \in \mathcal{P}^{\mathbb{R}_\sim}$ s.t. $\mathsf{ar}^{\mathbb{R}_\sim}(\Pi_n) = n$:

$$
C ::= \top \mid A \mid C \sqcap C \mid \exists R \mid \exists U_1, \dots, U_n.\Pi_n \qquad D ::= \bot \mid C \mid \forall U_1, \dots, U_n.\Pi_n
$$
$$
\mathfrak{a} ::= C \sqsubseteq D \mid R \sqsubseteq S\ .
$$

We assume (nested) conjunctions to be represented as sets, that is, they never contain duplicates, and the order of conjuncts is not important.

A *TBox* is a finite set of axioms. Let $A \in \mathsf{N_C}$, $a, b \in \mathsf{N_I}$, $r \in \mathsf{N_R}$, $U \in \mathsf{N_A}$, and $d \in \mathbb{R}$. An *ABox* is a set of *assertions*, which are of the forms $A(a)$, $r(a, b)$, and $U(a, d)$. A *knowledge base* (KB) is a tuple $\langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ an ABox.

**Example 1.** *Assume that $A, B_1, B_2 \in \mathsf{N_C}$, $r \in \mathsf{N_R}$, $U, U_1, U_2 \in \mathsf{N_A}$, and $b \in \mathsf{N_I}$. Then, let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a knowledge base with $\mathcal{T} = \{B_1 \sqsubseteq \exists r^-,\ A \sqsubseteq \exists U.{>_{3.5}},\ B_2 \sqsubseteq \forall U_1, U_2.{+_{10}}\}$ and $\mathcal{A} = \{A(b), B_2(b), U_1(b, 11), U_2(b, 21)\}$.*

The semantics of KBs is defined in terms of *interpretations*. An interpretation is a tuple $\mathcal{I} = \langle \Delta^\mathcal{I}, \cdot^\mathcal{I}, \mathbb{R}_\sim \rangle$, where $\Delta^\mathcal{I}$ is a set called the *domain*, $\cdot^\mathcal{I}$ is a function, and $\mathbb{R}_\sim$ is a concrete domain. The function $\cdot^\mathcal{I}$ maps every $a \in \mathsf{N_I}$ to an element $a^\mathcal{I} \in \Delta^\mathcal{I}$, every concept name $A$ to a set $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$, every role name $r$ to a relation $r^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$, and every attribute $U \in \mathsf{N_A}$ to a relation $U^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \mathbb{R}$. We require the domain $\Delta^\mathcal{I}$ to be disjoint with the concrete domain: $\Delta^\mathcal{I} \cap \mathbb{R} = \emptyset$. The interpretation function is extended to roles by setting $(r^-)^\mathcal{I} = (r^\mathcal{I})^-$, and to concepts by $(\top)^\mathcal{I} = \Delta^\mathcal{I}$, $(\bot)^\mathcal{I} = \emptyset$,

$$(C_1 \sqcap C_2)^\mathcal{I} = C_1^\mathcal{I} \cap C_2^\mathcal{I}, \qquad (\exists R)^\mathcal{I} = \{e \in \Delta^\mathcal{I} \mid \exists e' \in \Delta^\mathcal{I} : \langle e, e' \rangle \in R^\mathcal{I}\},$$
$$(\exists U_1, \ldots, U_n.\Pi)^\mathcal{I} = \{e \in \Delta^\mathcal{I} \mid \exists \langle e, d_1 \rangle \in U_1^\mathcal{I}, \ldots, \exists \langle e, d_n \rangle \in U_n^\mathcal{I} : \langle d_1, \ldots, d_n \rangle \in \Pi^{\mathbb{R}_\sim}\},$$
$$(\forall U_1, \ldots, U_n.\Pi)^\mathcal{I} = \{e \in \Delta^\mathcal{I} \mid \forall \langle e, d_1 \rangle \in U_1^\mathcal{I}, \ldots, \forall \langle e, d_n \rangle \in U_n^\mathcal{I} : \langle d_1, \ldots, d_n \rangle \in \Pi^{\mathbb{R}_\sim}\}.$$

Let $X$, $Y$ be concepts or roles. An interpretation $\mathcal{I}$ *satisfies an axiom* $X \sqsubseteq Y$ (in symbols $\mathcal{I} \models X \sqsubseteq Y$) iff $X^\mathcal{I} \subseteq Y^\mathcal{I}$. $\mathcal{I}$ *satisfies an assertion* $A(a)$ iff $a^\mathcal{I} \in A^\mathcal{I}$, $r(a, b)$ iff $\langle a^\mathcal{I}, b^\mathcal{I} \rangle \in r^\mathcal{I}$, and $U(a, d)$ iff $\langle a^\mathcal{I}, d \rangle \in U^\mathcal{I}$. $\mathcal{I}$ is a *model* of a KB (TBox) if it satisfies all axioms and assertions in it. Two TBoxes $\mathcal{T}$, $\mathcal{T}'$ are equivalent (in symbols $\mathcal{T} \equiv \mathcal{T}'$) if they have the same set of models. An axiom/assertion $\mathfrak{b}$ is *entailed* by a KB $\mathcal{K}$ (in symbols $\mathcal{K} \models \mathfrak{b}$) if $\mathcal{I} \models \mathfrak{b}$ for all models $\mathcal{I}$ of $\mathcal{K}$.

## 2.3 Conjunctive Queries for DL-Lite$_\sqcap(\mathbb{R}_\sim)$ KBs

Let $\mathsf{N_V}$ be a countably infinite set of *variables* pairwise disjoint with $\mathsf{N_C}$, $\mathsf{N_R}$, $\mathsf{N_A}$, and $\mathsf{N_I}$. Elements from $\mathsf{N_I} \cup \mathsf{N_V}$ are *abstract terms* and elements from $\mathbb{R} \cup \mathsf{N_V}$ are *concrete terms*. The union of abstract and concrete terms is called *terms*. Let $A \in \mathsf{N_C}$, $r \in \mathsf{N_R}$, $U \in \mathsf{N_A}$, $\Pi \in \mathcal{P}^{\mathbb{R}_\sim}$ with arity $n$, $t_a, t'_a$ be abstract terms and $t_{c_1}, \ldots, t_{c_n}$ be concrete terms. An *atom* is an expression of the forms $A(t_a)$, $r(t_a, t'_a)$, $=(t_a, t'_a)$, $U(t_a, t_c)$ or $\Pi(t_{c_1}, \ldots, t_{c_n})$. A *conjunctive query* (CQ) is an expression of the form $\phi = \exists x_1, \ldots, x_n.\alpha_1 \wedge \ldots \wedge \alpha_m$, where $x_1, \ldots, x_n \in \mathsf{N_V}$, and $\alpha_1, \ldots, \alpha_n$ are atoms. We denote terms in $\alpha$ ($/\phi$) by $\textbf{terms}(\alpha)$ ($/\textbf{terms}(\phi)$), and variables in $\alpha$ ($/\phi$) by $\textbf{var}(\alpha)$ ($/\textbf{var}(\phi)$). Variables in $\phi$ that are not bound by an existential quantifier are called *answer variables*. A *union of CQs* (UCQ) is a non-empty set of CQs, where each CQ has the same set of answer variables. We denote the set of answer variables of a UCQ $\Psi$ by $\mathsf{var}_{\text{ans}}(\Psi)$. A UCQ $\Psi$ is called *Boolean* if $\mathsf{var}_{\text{ans}}(\Psi) = \emptyset$. Given a UCQ $\Psi$, an *answer to* $\Psi$ is a mapping $\mathbf{a} : \mathsf{var}_{\text{ans}}(\Psi) \to \mathsf{N_I} \cup \mathbb{R}$, and we denote by $\mathbf{a}(\Psi)$ the Boolean UCQ obtained by replacing every answer variable $x$ by $\mathbf{a}(x)$. Answer variables and answers are defined accordingly for CQs.

Given an interpretation $\mathcal{I}$, a Boolean CQ $\phi$ is *satisfied by* $\mathcal{I}$ (in symbols $\mathcal{I} \models \phi$) if there exists a *homomorphism* $h : \textbf{terms}(\phi) \to \Delta^\mathcal{I} \cup \mathbb{R}$ s.t. for every $=(t_a, t'_a) \in \phi$ we have $h(t_a) = h(t'_a)$, for every $d \in \textbf{terms}(\phi) \cap \mathbb{R}$ we have $h(d) = d$, for every $a \in \textbf{terms}(\phi) \cap \mathsf{N_I}$ we have $h(a) = a^\mathcal{I}$, for every $A(t) \in \phi$ we have $h(t) \in A^\mathcal{I}$, for every $r(t_1, t_2) \in \phi$ we have $\langle h(t_1), h(t_2) \rangle \in r^\mathcal{I}$, for every $U(t_1, t_2) \in \phi$ we have $\langle h(t_1), h(t_2) \rangle \in U^\mathcal{I}$, and for every $\Pi(t_1, \ldots, t_n)$, we have $\langle h(t_1), \ldots, h(t_n) \rangle \in \Pi^{\mathbb{R}_\sim}$. We might then also write $\mathcal{I} \models h(\phi)$. A Boolean UCQ $\Phi$ is satisfied by $\mathcal{I}$ iff $\mathcal{I} \models \phi$ for some $\phi \in \Phi$. A Boolean UCQ is entailed by a KB $\mathcal{K}$ if it is satisfied in every model of $\mathcal{K}$. A Boolean CQ/UCQ $\Phi$ is unsatisfiable in a TBox $\mathcal{T}$ if for every model $\mathcal{I}$ of $\mathcal{T}$, we have $\mathcal{I} \not\models \Phi$. An answer $\mathsf{a}$ to a UCQ $\Psi$ is a *certain answer to* $\Psi$ if $\mathcal{K} \models \mathsf{a}(\Psi)$.

**Example 2.** *Let $\mathcal{K}$ be a KB as shown in Example 1. Let $\phi_1$ and $\phi_2$ be CQs s.t. $\phi_1 = \exists x, v.U(x,v)$, and $\phi_2 = \exists v_1, v_2.(U_1(x,v_1) \wedge U_2(x,v_2) \wedge +_{10}(v_1,v_2))$. We have $\mathsf{var}_{ans}(\phi_1) = \emptyset$, and $\mathsf{var}_{ans}(\phi_2) = \{x\}$. Furthermore, $\phi_1$ is Boolean and $\mathcal{K} \models \phi_1$. The answer $\mathsf{a}$ for $\phi_2$ with $\mathsf{a}(x) = b$ is a certain answer to $\phi_2$ in $\mathcal{K}$, since $\mathcal{K} \models \mathsf{a}(\phi_2)$.*

## 3 Overview of the Query Rewriting Method

Our aim is to develop a practical rewriting method for answering UCQs with concrete domain predicates over $\mathbb{R}_{\sim}$ w.r.t. DL-Lite$_{\sqcap}(\mathbb{R}_{\sim})$ ontologies. While in principle, a single rewriting step can be sufficient to enable query answering, it does not yield a practical algorithm that lends itself to implementation. This is mainly due to consequences that follow from the concrete domain alone, or from its combination with the TBox. For example, axioms of the form $C \sqsubseteq \forall U_1, U_2.+_d$ make both attributes $U_1$ and $U_2$ functional for instances of concept $C$. Consequences of this sort hold independently of the query and the data, and thus would need to be re-discovered for each answered UCQ in a classical rewriting system. A more efficient way is to compute these consequences once and reuse them. To this end, we present an approach that consists of two steps. The first one is a preprocessing step of the TBox $\mathcal{T}$, which is called *TBox saturation*, and the second is the rewriting of the query w.r.t. the saturated version of $\mathcal{T}$.

In the TBox saturation step, a set of *saturation rules* augments the TBox with additional axioms. The rewriting step is then similar to the classical one for DL-Lite [7], and employs a set of *rewriting rules*. Starting from some CQ $\phi$ in the input UCQ $\Phi$, and for every rewriting rule $\mathbf{R}$, if $\phi$ satisfies the premise of $\mathbf{R}$, and the side condition of $\mathbf{R}$ is also met, then the conclusion of $\mathbf{R}$, as a new CQ $\phi'$, is added to $\Phi$. This process is repeated for every CQ in $\Phi$ until a fixed-point is reached. As we present later on, the side conditions of the rewriting rules check for the existence of certain axioms that follow from $\mathcal{T}$, but are not necessarily present in $\mathcal{T}$. But since the query rewriting step is based on the saturated version of $\mathcal{T}$, a syntactic check suffices to inspect the satisfaction of these side conditions. In the extended version of this paper, we show that this method yields a sound and complete procedure for query rewriting for DL-Lite$_{\sqcap}(\mathbb{R}_{\sim})$.

## 4 TBox Saturation

We introduce the calculus for generating the saturated version of a given TBox $\mathcal{T}$, which is then used by the query rewriting procedure. Afterwards, we discuss properties of this calculus.

### 4.1 TBox Saturation Calculus

Our calculus consists of the rules shown in Figure 1. In these rules, $\Pi_1, \Pi_2 \in \mathcal{P}_{\mathbb{R}_{\sim}}$ are some predicates from $\mathbb{R}_{\sim}$, $\varpi \in \{<,=,>\}$ is a comparison operator from $\mathcal{P}^{\mathbb{R}_{\sim}}$, that together with a value $d$ gives rise to the unary predicate $\varpi_d$, and $\mathsf{Q} \in \{\forall, \exists\}$ is a quantifier. The preconditions are to be checked syntactically and the derived statements are added as axioms to the TBox.

The rules in the calculus are grouped according to the kind of inference they yield. The rules in $\mathbf{R_{init}}$ infer the straightforward properties of attributes. For example, rule $\mathbf{R_{init}}$-6 states that if an element has two attribute values with distance $d$, and both of these are (locally) functional, then all pairs of values of these attributes must have a distance $d$. The rules in $\mathbf{R}_{+}$ infer implicit distances between attribute values, since the binary predicate $+_d$ behaves additive for the real

$\mathbf{R_{init}}$-rules:

$$\frac{C \sqsubseteq \mathsf{Q}U_1, U_2.\Pi}{C \sqsubseteq \mathsf{Q}U_1.\top_{\mathbb{R}_\sim},\ C \sqsubseteq \mathsf{Q}U_2.\top_{\mathbb{R}_\sim}} \quad (1) \qquad \frac{C_1 \sqsubseteq \exists U.\Pi \quad C_2 \sqsubseteq \forall U, U.+_0}{C_1 \sqcap C_2 \sqsubseteq \forall U.\Pi} \quad (2)$$

$$\frac{C_1 \sqsubseteq \forall U.\Pi_1 \quad C_2 \sqsubseteq \exists U.\Pi_2}{C_1 \sqcap C_2 \sqsubseteq \exists U.\Pi_1} \quad (3) \qquad \frac{C_1 \sqsubseteq \forall U_1, U_2.+_d}{\exists U_1.\top_{\mathbb{R}_\sim} \sqcap \exists U_2.\top_{\mathbb{R}_\sim} \sqcap C_1 \sqsubseteq \exists U_1, U_2.+_d} \quad (4)$$

$$\frac{C \sqsubseteq \mathsf{Q}U_1, U_2.+_d}{C \sqsubseteq \mathsf{Q}U_2, U_1.+_{-d}} \quad (5) \qquad \frac{C_1 \sqsubseteq \exists U_1, U_2.+_d \quad C_2 \sqsubseteq \forall U_1, U_1.+_0 \quad C_3 \sqsubseteq \forall U_2, U_2.+_0}{C_1 \sqcap C_2 \sqcap C_3 \sqsubseteq \forall U_1, U_2.+_d} \quad (6)$$

$\mathbf{R}_+$-rules:

$$\frac{C_1 \sqsubseteq \exists U_1, U_2.+_{d_1} \qquad C_2 \sqsubseteq \exists U_2, U_3.+_{d_2} \qquad C_3 \sqsubseteq \forall U_2, U_2.+_0}{C_1 \sqcap C_2 \sqcap C_3 \sqsubseteq \exists U_1, U_3.+_{(d_1+d_2)}} \quad (1)$$

$$\frac{C_1 \sqsubseteq \forall U_1, U_2.+_{d_1} \quad C_2 \sqsubseteq \forall U_2, U_3.+_{d_2}}{\exists U_2.\top_{\mathbb{R}_\sim} \sqcap C_1 \sqcap C_2 \sqsubseteq \forall U_1, U_3.+_{(d_1+d_2)}} \quad (2) \qquad \frac{C_1 \sqsubseteq \forall U_1, U_2.+_{d_1} \quad C_2 \sqsubseteq \exists U_2, U_3.+_{d_2}}{\exists U_1.\top_{\mathbb{R}_\sim} \sqcap C_1 \sqcap C_2 \sqsubseteq \exists U_1, U_3.+_{(d_1+d_2)}} \quad (3)$$

$\mathbf{R}_\varpi$-rules:

$$\frac{C_1 \sqsubseteq \forall U_1, U_2.+_{d_1} \qquad C_2 \sqsubseteq \exists U_1.\varpi_{d_2} \qquad C_3 \sqsubseteq \exists U_2.\Pi}{C_1 \sqcap C_2 \sqcap C_3 \sqsubseteq \exists U_2.\varpi_{(d_1+d_2)}} \quad (1)$$

$$\frac{C_1 \sqsubseteq \forall U_1, U_2.+_{d_1} \qquad C_2 \sqsubseteq \exists U_1.\varpi_{d_2}}{C_1 \sqcap C_2 \sqsubseteq \forall U_2.\varpi_{(d_1+d_2)}} \quad (2) \qquad \frac{C_1 \sqsubseteq \exists U_1, U_2.+_{d_1} \qquad C_2 \sqsubseteq \forall U_1.\varpi_{d_2}}{C_1 \sqcap C_2 \sqsubseteq \exists U_2.\varpi_{(d_1+d_2)}} \quad (3)$$

$\mathbf{R}_\perp$-rules:

$$\frac{C_1 \sqsubseteq D_1 \qquad C_2 \sqsubseteq D_2}{C_1 \sqcap C_2 \sqsubseteq \perp} \qquad \text{provided } \models D_1 \sqcap D_2 \sqsubseteq \perp \quad (1)$$

$$\frac{C_1 \sqsubseteq D_1 \qquad C_2 \sqsubseteq D_2}{C_1 \sqcap C_2 \sqsubseteq \forall U.\perp_{\mathbb{R}_\sim}\ /\ \forall U_1, U_2.\perp_{\mathbb{R}_\sim}} \qquad \text{provided } \models D_1 \sqcap D_2 \sqsubseteq \forall U.\perp_{\mathbb{R}_\sim}\ /\ \forall U_1, U_2.\perp_{\mathbb{R}_\sim} \quad (2)$$

$$\frac{C_1 \sqcap \exists U.\top_{\mathbb{R}_\sim} \sqsubseteq D}{C_1 \sqcap C_2 \sqsubseteq D} \qquad \text{provided } C_2 \sqsubseteq \exists U.\Pi \quad (3)$$

Figure 1: TBox saturation rules.

numbers, and distances can simply be propagated down (/up) the number line. Rules in $\mathbf{R}_\varpi$ lead to the inference of an attribute value based on the following: if the distance between two attribute successors and the value of one of them are known, then the value of the other attribute successor can be inferred. Lastly, the rules in $\mathbf{R}_\perp$ lead to axioms stating which concepts cannot have certain attribute successors or which concepts are unsatisfiable. Observe that the saturation rules can refer to (the presence of) data while staying data-independent. This is achieved by the use of $\exists U.\top_{\mathbb{R}_\sim}$ in the left-hand side of the inferred statements.

## 4.2 Properties of the Saturation Calculus

In Algorithm 1, the saturation rules are used to infer all axioms from the TBox which are relevant to our rewriting procedure. The relevance of these axioms is determined mainly by *rewritability* and *termination*. To illustrate relevance of axioms for rewritability, let us take the following case as an example. Assume we are given the TBox $\mathcal{T} = \{C_2 \sqsubseteq \exists U_1.{=_3},\ \top \sqsubseteq \exists U_2.\top_{\mathbb{R}_\sim},\ C_1 \sqsubseteq \forall U_1, U_2.{+_2}\}$, the ABox $\mathcal{A} = \{C_1(b),\ C_2(b)\}$, and the CQ $\phi = \exists v.(U_2(x,v) \wedge {=_v}(5))$. It is easy to see that $\langle \mathcal{T}, \mathcal{A}\rangle \models \phi$, but $\langle \emptyset, \mathcal{A}\rangle \not\models \phi$. In order to be able to rewrite $\phi$ into $\Phi$ s.t. $\langle \emptyset, \mathcal{A}\rangle \models \Phi$, we need some axiom $\mathfrak{a}$ in $\mathcal{T}$ of the form $C_1 \sqcap C_2 \sqsubseteq \exists U_2.{=_5}$. Thus, $\mathfrak{a}$ is a *relevant* axiom from the *rewritability* aspect, and therefore, a rule to generate such an axiom, from such a TBox, is needed. In this example, this rule is $\mathbf{R}_\varpi$-1. Let us consider $\phi' = \exists v.(U_2(x,v))$, where $\mathfrak{a}_1 = (C_1 \sqcap C_2 \sqsubseteq \exists U_2.\top_{\mathbb{R}_\sim})$ is needed to get the rewriting of $\phi$, but actually $\mathfrak{a}_1$ is not a relevant axiom because its effect is already covered, since $(C_1 \sqcap C_2 \sqsubseteq \exists U_2.{=_5}) \models (C_1 \sqcap C_2 \sqsubseteq \exists U_2.\top_{\mathbb{R}_\sim})$. This type of entailments is handled by the rewriting rules from Section 5.

The other criterion for relevance is *termination.* The interaction between the rules, or even between the conclusion and the premise of the same rule, enables infinitely many rule applications. Axioms inferred using the $\mathbf{R}_\bot$ can be utilised to prevent such behaviour: note that an axiom of the form $C \sqsubseteq \forall U.\bot_{\mathbb{R}_\sim}$ makes all other axioms of the form $C \sqsubseteq \forall U.\Pi$ superfluous, which can be used to limit the number of inferred axioms to be kept. Therefore the $\mathbf{R}_\bot$ rules are applied preferred to any other rule, see Algorithm 1.

Let $\mathcal{T}$ be a TBox, $\mathbf{R}$ a saturation rule from Figure 1, and $\mathfrak{a}$ an axiom. The axiom $\mathfrak{a}$ is *derivable by* $\mathbf{R}$ *from* $\mathcal{T}$ (in symbols $\mathcal{T} \vdash_{\mathbf{R}} \mathfrak{a}$) iff the premise(s) of $\mathbf{R}$ occur in $\mathcal{T}$ and $\mathfrak{a}$ is of the form of the conclusion of $\mathbf{R}$. $\mathcal{T} \vdash \mathfrak{a}$ denotes that $\mathfrak{a}$ *is derivable from* $\mathcal{T}$ using any saturation rule in the calculus. We show in the extended version of the paper, that the calculus yields a sound TBox saturation procedure.

**Lemma 1** (Soundness)**.** *Let $\mathcal{T}$ be a TBox and $\mathfrak{a}$ be an axiom. Then, $\mathcal{T} \vdash \mathfrak{a}$ only if $\mathcal{T} \models \mathfrak{a}$.*

To ensure termination of the TBox saturation process, we refer to "superfluous axioms".

**Definition 2** (Redundant axiom)**.** *Let $\mathcal{T}$ be a TBox and $\mathfrak{a}_1, \mathfrak{a}_2 \in \mathcal{T}$. Then, $\mathfrak{a}_2$* is redundant *to $\mathfrak{a}_1$ w.r.t. $\mathcal{T}$ if at least one of the following conditions is satisfied:*

1. *$\mathfrak{a}_1$ is of the form $C \sqsubseteq D$, and $\mathfrak{a}_2$ is of the form $C \sqcap C' \sqsubseteq D$;*
2. *$\mathfrak{a}_1$ is of the form $C \sqsubseteq \forall U.\bot_{\mathbb{R}_\sim}$, and $\mathfrak{a}_2$ is of the form $C \sqcap C' \sqsubseteq \forall U.\Pi$; or*
3. *$\mathfrak{a}_1$ is of the form $C \sqsubseteq \forall U_1, U_2.\bot_{\mathbb{R}_\sim}$, and $\mathfrak{a}_2$ is of the form $C \sqcap C' \sqsubseteq \forall U_1, U_2.{+_d}$.*

*The axiom $\mathfrak{a}_2 \in \mathcal{T}$ is a* redundant axiom in $\mathcal{T}$ *iff there exists some $\mathfrak{a}_1 \in \mathcal{T}$ such that $\mathfrak{a}_2$* is redundant to $\mathfrak{a}_1$ *w.r.t. $\mathcal{T}$.*

From this definition, it follows that for any TBox $\mathcal{T}$, we have $\mathcal{T} \equiv \mathcal{T} \setminus \{\mathfrak{a}_2\}$ where $\mathfrak{a}_1, \mathfrak{a}_2 \in \mathcal{T}$, and $\mathfrak{a}_1$ makes $\mathfrak{a}_2$ redundant in $\mathcal{T}$. Therefore, a refinement function of $\mathcal{T}$ can be defined as $\mathsf{refine}(\mathcal{T}) = \{\mathfrak{a} \in \mathcal{T} \mid \mathfrak{a} \text{ is not redundant in } \mathcal{T} \setminus \{\mathfrak{a}\}\}$. Algorithm 1 specifies the computation of the saturated TBox (in symbols: *saturate*($\mathcal{T}$)). In the extended version of the paper, we prove that for any TBox $\mathcal{T}$, and due to redundancy elimination, Algorithm 1 terminates on any input.

**Algorithm 1:** Computation of *saturate*($\mathcal{T}$)

```
Input: TBox $\mathcal{T}$.
while $\mathcal{T} \vdash \mathfrak{a}$ for some $\mathfrak{a}$ and $\mathfrak{a}$ not redundant in $\mathcal{T}$ do
    while $\mathcal{T} \vdash_{\mathbf{R}_\bot} \mathfrak{a}'$ for some $\mathfrak{a}'$ and $\mathfrak{a}'$ not redundant in $\mathcal{T}$ do
        set $\mathcal{T} := \mathsf{refine}(\mathcal{T} \cup \{\mathfrak{a}'\})$.
    set $\mathcal{T} := \mathsf{refine}(\mathcal{T} \cup \{\mathfrak{a}\})$.
return $\mathcal{T}$;
```

**Theorem 3** (Termination of saturation)**.** *Algorithm 1 always terminates.*

In the following, we show an example of the saturation process of a given TBox $\mathcal{T}$.

**Example 3.** *Consider the TBox $\mathcal{T} = \{A \sqsubseteq \exists U_1.{=_{2.6}},\ B \sqsubseteq \forall U_2, U_1.{+_{0.4}}\}$. We show the computation of some axioms in saturate$(\mathcal{T})$, by applying the rules from Figure 1 as follows:*

$$\begin{aligned}\mathcal{T} \cup \{ &\xrightarrow{\mathbf{R}_{init}-5} B \sqsubseteq \forall U_1, U_2.{+_{-0.4}}\ ,\ \xrightarrow{\mathbf{R}_{init}-4} \exists U_1.\top_{\mathbb{R}_\sim} \sqcap \exists U_2.\top_{\mathbb{R}_\sim} \sqcap B \sqsubseteq \exists U_2, U_1.{+_{0.4}}\ ,\\ &\xrightarrow{\mathbf{R}_{init}-5} \exists U_1.\top_{\mathbb{R}_\sim} \sqcap \exists U_2.\top_{\mathbb{R}_\sim} \sqcap B \sqsubseteq \exists U_1, U_2.{+_{-0.4}}\ ,\\ &\xrightarrow{\mathbf{R}_{+}-2} \exists U_2.\top_{\mathbb{R}_\sim} \sqcap B \sqsubseteq \forall U_1, U_1.{+_0}\ ,\ \exists U_1.\top_{\mathbb{R}_\sim} \sqcap B \sqsubseteq \forall U_2, U_2.{+_0}\ ,\\ &\xrightarrow{\mathbf{R}_{init}-2} \exists U_2.\top_{\mathbb{R}_\sim} \sqcap A \sqcap B \sqsubseteq \forall U_1.{=_{2.6}}\ ,\ \xrightarrow{\mathbf{R}_{\varpi}-1} \exists U_2.\top_{\mathbb{R}_\sim} \sqcap A \sqcap B \sqsubseteq \forall U_2.{=_{2.2}}\ ,\ \ldots\}.\end{aligned}$$

## 5 Query Rewriting

This section presents the computation procedure for rewriting a given UCQ $\Phi$ w.r.t. a saturated TBox. The idea is to apply a set of rules on $\Phi$, so that the union over the resulting set of CQs $\Phi'$ has exactly those certain answers over $\langle \emptyset, \mathcal{A}\rangle$ as $\Phi$ has over $\langle \mathcal{T}, \mathcal{A}\rangle$. We make the following *simplifying assumptions* on CQs $\phi$ in $\Phi$: *i)* for every atom of the form $=(t_a, t'_a) \in \phi$, $t_a \in \mathsf{var}_{\mathrm{ans}}(\phi)$, and *ii)* $\mathbf{terms}(\phi) \cap \mathbb{R} = \emptyset$. These assumptions are w.l.o.g. since for *i)*, if $=(t_a, t'_a) \in \phi$ and $t_a \notin \mathsf{var}_{\mathrm{ans}}(\phi)$, we can replace $t_a$ by $t'_a$ exhaustively in the query, and equality between different constants that would make the query unsatisfiable can be easily detected; and for *ii)*, since we can replace every real number $d \in \mathbf{terms}(\phi)$ by a variable $v_d$, for which we add the atom $=_d(v_d)$. To keep queries in that form, we use a slightly non-standard notion of substitutions.

**Definition 4** (Substitution)**.** *Let $\phi$ be a CQ, $\alpha$ an atom s.t. $\alpha \in \phi$. A* substitution *on $\phi$ is a function $\sigma : \mathbf{terms}(\phi) \to \mathsf{N_V} \cup \mathsf{N_I}$, with the additional requirement that $\sigma(t) = t$ if $t \in \mathsf{N_I}$. The result of applying $\sigma$ on atom $\alpha$ (in symbols: $\alpha\sigma$) is an atom obtained by replacing every term $t \in \mathbf{terms}(\alpha)$ by $\sigma(t)$. The result of applying $\sigma$ on CQ $\phi$ (in symbols: $\phi\sigma$) is obtained from $\phi$ by replacing every atom $\alpha \in \phi$ of the form $=(t_1, t_2)$ by $=(t_1, \sigma(t_2))$ and every other atom $\alpha$ by $\alpha\sigma$, and adding an atom $=(x, t)$ for every answer variable $x$ s.t. $\sigma(x) = t \neq x$.*

Note the special treatment of atoms of the form $=(t_1, t_2)$: by assumption *i)*, these are only used to express equality of answer variables with other terms. Definition 4 ensures that we can substitute answer variables in the remaining query without affecting the answers of the query.

Before we discuss the rewriting rules, we need to address the syntactic mismatch between queries and axioms. Specifically, the rewriting rules may operate on roles or complex concepts as they can refer to axioms. Since the syntax of CQs does not admit these, we employ equivalences that "bridge" this gap between query and rules. The idea is that the syntactic matching of rules to a CQ is done modulo these equivalences. Let $\phi$ be a CQ, $\{x, y\} \subseteq \mathsf{N_V}$, and $X$ be a role or a concept allowed on the left-hand side of an axiom. If $X$ is a role and $X = r^-$, then $(\phi \wedge r^-(x, y)) \equiv (\phi \wedge r(y, x))$. Let $v, w \in \mathsf{N_V} \setminus \mathbf{var}(\phi)$. If $X$ is a concept, then, depending on the structure of $X$, the CQ $\widehat{\phi} = (\phi \wedge X(x))$ is equivalent to:

- $\phi \wedge C_1(x) \wedge C_2(x)$, if $X = C_1 \sqcap C_2$;
- $\phi \wedge U(x, v) \wedge \Pi(v)$, if $X = \exists U.\Pi$; but it is $\phi \wedge U(x, w)$, if $C = \exists U.\top_{\mathbb{R}_\sim}$;
- $\phi \wedge U_1(x, v) \wedge U_2(x, w) \wedge \Pi(v, w)$, if $X = \exists U_1, U_2.\Pi$;
- $\phi \wedge A(x)$, if $X = A \in \mathsf{N_C}$;
- $\phi$, if $X = \top$;
- $\phi \wedge R(x, w)$, if $X = \exists R$;

| | | | | | |
|---|---|---|---|---|---|
| **RR1** | $\frac{\phi \wedge X(\vec{t})}{\phi \wedge Y(\vec{t})}$ | $Y \sqsubseteq X'$, $\models X' \sqsubseteq X$ | **RR2** | $\frac{\phi \wedge \Pi_1(v)}{\phi \wedge C(x) \wedge U(x, v)}$ | $C \sqsubseteq \forall U.\Pi_2$, $\Pi_2 \models \Pi_1$ |

| | | |
|---|---|---|
| **RR3** | $\frac{\phi \wedge +_d(v_1, v_2)}{\phi \wedge C(x) \wedge U_1(x, v_1) \wedge U_2(x, v_2)}$ | $C \sqsubseteq \forall U_1, U_2.+_d$ |
| **RR4** | $\frac{\phi \wedge U(x, v)}{\phi \wedge C(x) \wedge =_d(v)}$ | $C \sqsubseteq \exists U.=_d$ |
| **RR5** | $\frac{\phi \wedge U_1(x, v_1) \wedge U_2(y, v_2) \wedge +_d(v_1, v_2)}{\phi[y \mapsto x] \wedge C(x) \wedge U_1(x, v_1)}$ | $C \sqsubseteq \exists U_1, U_2.+_d$, $v_2 \notin \mathsf{var}(\phi)$ |
| **RR6** | $\frac{\phi \wedge U_1(x, v_1) \wedge \varpi_{d_1}(v_1)}{\phi \wedge C(x) \wedge U_1(x, v_1) \wedge U_2(x, v_2) \wedge \varpi_{d_1+d_2}(v_2)}$ | $C \sqsubseteq \forall U_1, U_2.+_{d_2}$ |
| **RR7** | $\frac{\phi \wedge U_1(x, v_1) \wedge \varpi_{d_1}(v_1)}{\phi \wedge (C_1 \sqcap C_2)(x) \wedge U_2(x, v_2) \wedge +_{d_2}(v_1, v_2) \wedge \varpi_{d_1+d_2}(v_2)}$ | $C_1 \sqsubseteq \forall U_2, U_2.+_0$, $C_2 \sqsubseteq \exists U_1, U_2.+_{d_2}$ |
| **RR8** | $\frac{\phi \wedge U_1(x, v_1) \wedge +_{d_1}(v_2, v_1)}{\phi \wedge C(x) \wedge U_1(x, v_1) \wedge U_2(x, v_3) \wedge +_{d_1+d_2}(v_2, v_3)}$ | $C \sqsubseteq \forall U_1, U_2.+_{d_2}$ |
| **RR9** | $\frac{\phi \wedge U_1(x, v_2) \wedge +_{d_1}(v_1, v_2)}{\phi \wedge (C_1 \sqcap C_2)(x) \wedge U_2(x, v_3) \wedge +_{d_1+d_2}(v_1, v_3) \wedge +_{d_2}(v_2, v_3)}$ | $C_1 \sqsubseteq \forall U_2, U_2.+_0$, $C_2 \sqsubseteq \exists U_1, U_2.+_{d_2}$ |

Figure 2: Query rewriting rules dependant on the TBox. (If a variable $x$ occurs only in the conclusion of a rule, we assume $\phi$ contains an atom of the form $U'(x, v')$.)

| | | | |
|---|---|---|---|
| **RC1** | $\frac{\phi \wedge +_{d_1}(v_1, v_2) \wedge +_{d_2}(v_2, v_3)}{\phi \wedge +_{d_1}(v_1, v_2) \wedge +_{(d_1+d_2)}(v_1, v_3)}$ | **RC2** | $\frac{\phi \wedge +_d(v_1, v_2)}{\phi \wedge +_{(-d)}(v_2, v_1)}$ |
| **RC3** | $\frac{\phi \wedge =_{d_1}(v_1) \wedge =_{d_2}(v_2)}{\phi \wedge =_{d_1}(v_1) \wedge +_{(d_2-d_1)}(v_1, v_2)}$ | **RC4** | $\frac{\phi \wedge =_{d_1}(v_1) \wedge +_{d_2}(v_1, v_2)}{\phi \wedge =_{d_1}(v_1) \wedge =_{(d_1+d_2)}(v_2)}$ |
| **RC5** | $\frac{\phi \wedge +_{d_2}(v_1, v_2) \wedge \varpi_{d_1}(v_1)}{\phi \wedge +_{d_2}(v_1, v_2) \wedge \varpi_{(d_1+d_2)}(v_2)}$ | **RC6** | $\frac{\phi \wedge \Pi_1(v) \wedge \Pi_2(v)}{\phi \wedge \Pi_1(v)}, \Pi_1 \models \Pi_2$ |
| **RC7** | $\frac{\phi \wedge U_1(x_1, v_1) \wedge U_2(x_2, v_1)}{\phi \wedge U_1(x_1, v_1) \wedge U_2(x_2, v_2) \wedge +_0(v_1, v_2)}$ | **RC8** | $\frac{\phi \wedge \varpi_d(w)}{\phi}$ |
| **RC9** | $\frac{\phi \wedge U(x, v) \wedge U(x, w) \wedge +_0(v_1, w)}{\phi \wedge U(x, v)}$ | **RC10** | $\frac{\phi \wedge +_d(v, w)}{\phi}$ |

Figure 3: Query rewriting rules independant on the TBox. (Here, $w$ denotes a unique non-distinguished variable.)

The rewriting rules are grouped into TBox-dependent rules (see Figure 2) and TBox-independent rules (see Figure 3). **RR1** is the standard rewriting rule for DL-Lite, and **RR2** and **RR3** are the variant dealing with universal restrictions, already used in [4]. **RR5** corresponds to a special case of **RR1** in which an additional substitution step is required. The main function of most rules in Figure 3 is to reformulate the concrete domain expressions in the query in an equivalence preserving way. While the TBox saturation already computes some inferences between concrete domain predicates, not every possible combination of concrete domain predicates that may occur in a query can be treated oblivious of the query. It is thus possible that the query contains concrete domain predicates saturation has not considered yet, but which can be transformed into the required form. This is done by Rules **RC1** to **RC5** and **RC7**. The purpose of other rules is to reduce the number of occurrences of a variable, to make other rules applicable. This is the main motivation behind **RR4**, **RC6**, **RC9** and **RC10**, and is also achieved by **RR6** to **RR9**. Note that we have to be careful here not to lose the "link" of any variable occurring in the rest of the query: if we would drop a variable occurring elsewhere, we would allow for additional matches not covered by the original query. In [4], shared variables in the query are eliminated using a special *splitting-rule*, which splits shared occurrences of a variable by assigning a fixed value to them, where the set of values is determined based on the TBox and the ABox. Since we want to obtain a data-independent and goal-oriented procedure, our rewriting procedure does not use a splitting rule.

However, in order to achieve full data-independence, a bit more has to be done. Note that the fillers of concrete domain attributes can be determined by both: numbers occurring in the data and axioms in the TBox. If a predicate in the query refers to an attribute filler implicit in the data, we may need to "push" the concrete domain predicates in the query towards those attribute fillers explicit in the data. This is the purpose of rules **RR6** to **RR9**. Note that these rules, similar to some of the TBox saturation rules, may make use of local functionalities of an attribute expressed by an axiom of the form $C \sqsubseteq \forall U, U.+_0$.

To obtain a rewriting procedure that is both complete and terminating, two problems have to be addressed. First, the rules need not be applicable to a query, as some rules require certain variables to occur only once or twice in the query. If a CQ contains a variable multiple times, it is often possible to reduce the number of occurrences by applying appropriate substitutions. Second, termination of the rewriting has to be ensured. Note that rules **RR6** to **RR9** rely on predicate atoms of the form $\varpi_d(v)$ (or $+_d(v, v')$) in their precondition and generate predicate atoms with same kind of predicate, but with a new value for $d$ and thus with a new predicate and with different variables. This generation process can continue, but it generates only redundant queries. We adress the two problems in the following.

**Achieving applicability of rules** can require to unify soame variables, which is usually achieved by applying substitutions. From a single CQ, a set of many CQs can be derived from the same CQ by such substitutions. One can easily see that not all possible substitutions are relevant here, but only those that unify different atoms. We make this intuition formal. Let $A = \{\alpha_1, \ldots, \alpha_n\}$ be a set of atoms. Set $A$ *unifies* (to a singleton) if there exists a substitution $\sigma$ s.t. $\alpha_1\sigma = \ldots = \alpha_n\sigma$. We then call $\sigma$ a *unifier of* $A$. For each such set, we select a *most general unifier* $\mathsf{mgu}(A)$, which is defined as a unifier $\sigma$ of $A$ s.t. for every other unifier $\sigma'$ of $A$, there exists some substitution $\sigma''$ s.t. $\sigma \circ \sigma'' = \sigma'$. We define the set of all CQs that can be obtained by unifying any set of atoms in the CQ $\phi$ as $\mathsf{reduce}(\phi) = \{\phi\sigma \mid \sigma = \mathsf{mgu}(A), A \subseteq \phi, A \text{ unifies}\}$.

Let $\Phi$ be a UCQ. For every $\phi \in \Phi$, due to our notion of redundancy that is to be defined next, every $\phi' \in \mathsf{reduce}(\phi)$ is redundant. This is why $\mathsf{reduce}(\phi)$ is not defined as a rewriting rule, but as a separate procedure.

**Achieving termination of rule application** depends on limiting the number of new predicates introduced by rules, and on limiting the number of variables introduced. The latter effect can be remedied by avoiding redundant queries in the query set.

**Definition 5** (Redundant query). *Given two CQs $\phi_1$ and $\phi_2$, $\phi_2$ is* redundant *to $\phi_1$, iff there exists a substitution $\sigma$ s.t. $\phi_1\sigma \subseteq \phi_2$. CQ $\phi_2$ is redundant in a UCQ $\Phi$ if there exists $\phi_1 \in \Phi$ to which $\phi_2$ is redundant.*

By Lemma 6, removing redundant CQs from $\Phi$ does not affect the entailment of $\Phi$.

**Lemma 6** (Query redundancy elimination). *Let $\Phi$ be a Boolean UCQ, and $\phi_1, \phi_2 \in \Phi$ s.t. $\phi_2$ is redundant to $\phi_1$. It holds for every interpretation $\mathcal{I}$, that $\mathcal{I} \models \Phi$ iff $\mathcal{I} \models \Phi \setminus \{\phi_2\}$.*

*Proof.* Let $\mathcal{I}$ be an arbitrary interpretation s.t. $\mathcal{I} \models \Phi \setminus \{\phi_2\}$. By the definition of entailment of UCQs, and since $\Phi \setminus \{\phi_2\} \subsetneq \Phi$, we have $\mathcal{I} \models \Phi$. For the reverse direction, let $\mathcal{I}$ be an arbitrary interpretation, and assume $\mathcal{I} \models \Phi$ and $\mathcal{I} \not\models \Phi \setminus \{\phi_2\}$. This means two things. First, $\mathcal{I} \models \phi_2$, and therefore there exists a homomorphism $h : \textsf{terms}(\phi_2) \to \Delta^{\mathcal{I}} \cup \mathbb{R}$. Second, $\mathcal{I} \not\models \phi_1$. But since $\phi_2$ is redundant to $\phi_1$, there exists a substitution $\sigma$ s.t. $\phi_1\sigma \subseteq \phi_2$. We know that $\sigma$ is a mapping from terms in $\phi_1$ to terms in $\phi_2$, and $h$ is a mapping from terms in $\phi_2$ to elements in $\Delta^{\mathcal{I}} \cup \mathbb{R}$. Therefore, we can construct a mapping $h' = \sigma \circ h$ s.t. $h' : \textsf{terms}(\phi_1) \to \Delta^{\mathcal{I}} \cup \mathbb{R}$. The function $h'$ is indeed a homomorphism from terms of $\phi_1$ into $\mathcal{I}$. Hence $\mathcal{I} \models \phi_1$, and consequently, $\mathcal{I} \models \Phi \setminus \{\phi_2\}$, which contradicts the original assumption. □

**The algorithm to compute the rewriting** of a UCQ $\Phi$, written as *rew*($\Phi$), uses the rewriting rules shown in Figures 2 and 3, where $X$ and $Y$ are both either roles or concepts. The complete UCQ rewriting is depicted in Algorithm 2.

**Example 4.** *Let $\mathcal{T} = \{\mathfrak{a}_1 = A \sqsubseteq \exists U_3, U_1.+_3,\ \mathfrak{a}_2 = A \sqsubseteq \forall U_1, U_2.+_1,\ \mathfrak{a}_3 = B \sqsubseteq \exists U_2.\top_{\mathbb{R}_\sim}\}$ be a TBox, $\mathcal{A} = \{A(c), B(c), U_1(c, 10)\}$ an ABox, and $\phi = \exists v_3.(U_3(x, v_3) \land >_5(v_3))$ a CQ. Then, $\mathbf{a}$ with $\mathbf{a}(x) = c$ is a certain answer to $\phi$ in $\langle \mathcal{T}, \mathcal{A} \rangle$. In the following, we show how it is obtained. First, we compute saturate($\mathcal{T}$), which contains $\mathfrak{a}_4 = \exists U_2.\top_{\mathbb{R}_\sim} \sqcap A \sqsubseteq \forall U_1, U_1.+_0$, among other axioms. This axiom is derived by applying* $\mathbf{R_{init}}$*-5 on $\mathfrak{a}_2$, and then* $\mathbf{R_+}$*-2 on the result and $\mathfrak{a}_2$. To obtain the CQ for which $\mathbf{a}$ is a certain answer in $\langle \emptyset, \mathcal{A} \rangle$, we compute rew($\phi$), which consists of $\phi$ and the following queries:*

$$\begin{aligned}
&\overset{RR7(\mathfrak{a}_1,\mathfrak{a}_4)}{\rightsquigarrow} && A(x) \land U_1(x, v_1) \land U_2(x, v_2) \land +_3(v_3, v_1) \land >_8(v_1) \\
&\overset{RC10}{\rightsquigarrow} && A(x) \land U_1(x, v_1) \land U_2(x, v_2) \land >_8(v_1) \\
&\overset{RR1(\mathfrak{a}_3)}{\rightsquigarrow} && A(x) \land U_1(x, v_1) \land B(x) \land >_8(v_1) \qquad = \phi'.
\end{aligned}$$

*In $\langle \emptyset, \mathcal{A} \rangle$, $\phi'$ has a match, and therefore $\mathbf{a}$ is a certain answer to rew($\phi$).*

Let $\mathcal{T}$ be a saturated TBox, $\phi$ a CQ, and **RR** some rewriting rule. In Algorithm 2, we denote by $\phi \overset{\mathbf{RR},\mathcal{T}}{\rightsquigarrow} \phi'$ a rewriting step w.r.t. $\mathcal{T}$ s.t. some atom(s) in $\phi$ satisfy the premise(s) of **RR**, and the resulting query $\phi'$ is in the form of the conclusion of **RR**.

In order to prove termination of the algorithm, we need to show that the number of generated rewritings (CQs) is bounded. Actually, whether a rewriting rule depends on some axiom $\mathfrak{a} \in \mathcal{T}$ or not, it is in theory possible to generate an infinite number of CQs using our rewriting rules, unless we restrict the addition of new CQs appropriately. The reason behind this is that some of these rules may introduce an unbounded number of variables and atoms with new concrete domain predicates. We show, in the extended version of the paper, that queries which would trigger an unbounded application of rewriting rules are indeed either redundant or unsatisfiable. Thus, eliminating such queries prevents the generation of an unbounded number of CQs.

**Algorithm 2:** Computation of $rew(\Phi)$

```
Input: UCQ $\Phi$, saturated TBox $\mathcal{T}$.
for $\phi \in \Phi$ do
    for $\phi_r \in reduce(\phi)$ and $\psi$ such that $\phi_r \overset{\mathbf{RR},\mathcal{T}}{\rightsquigarrow} \psi$ do
        if $\psi$ is satisfiable w.r.t. $\mathcal{T}$ and not redundant in $\Phi$ then
            set $\Phi := \{\psi' \in \Phi \mid \psi'$ is not redundant to $\psi\} \cup \{\psi\}$;
return $\Phi$;
```

**Theorem 7** (Termination of rewriting)**.** *Algorithm 2 always terminates.*

In the extended version of the paper, we show that the obtained rewriting yields a sound and complete query rewriting procedure. We start by proving that for any Boolean CQ $\phi$, $\langle\mathcal{T},\mathcal{A}\rangle \models \phi$ iff $\langle\emptyset,\mathcal{A}\rangle \models rew(\phi)$. Once this is proven, and because of the definition of entailment of UCQs, we obtain the following theorem.

**Theorem 8** (Soundness and completeness of rewriting)**.** *Let $\mathcal{T}$ be a saturated TBox and $\mathcal{A}$ be an ABox s.t. $\mathcal{K} = \langle\mathcal{T},\mathcal{A}\rangle$ is satisfiable. Then, for any UCQ $\Phi$, and any answer $\mathbf{a}$ to $\Phi$, $\mathbf{a}$ is a certain answer to $\Phi$ in $\mathcal{K}$ iff $\mathbf{a}$ is a certain answer to $rew(\Phi)$ in $\langle\emptyset,\mathcal{A}\rangle$.*

## 6 Conclusion

In this paper, we have considered extensions of DL-Lite with concrete domains over $\mathbb{R}$. In this setting, we have presented a query rewriting approach that handles not only unary concrete domain predicates, but also binary ones. The key idea is that entailments that are caused by the TBox and the functionality of the cr-rewritable concrete domains employed here, are computed beforehand and are added to the TBox during saturation. Although this step can be costly, it only needs to be performed once (for all queries) and can be done "off-line", before query execution. Furthermore, we have shown that our approach yields a sound, complete, and terminating query rewriting procedure. This procedure is, in contrast to earlier approaches [4], data-independent. The latter is crucial for gaining practicality on larger data sets, and our method allows for a more goal-oriented computation of rewritings than the other approaches. In order to support this claim, we are currently working on an implementation of our approach, where the resulting UCQ is translated into a union of SQL queries which then can be answered using any RDBMS. We plan to use the rewritings not only on classical data sets, but also on probabilistic data in which concrete domain values are characterised by continuous probability distributions. OBQA for this setting has been theoretically investigated in [6].

As for future work, there are various directions we are considering. First, we would like to investigate the possibility of extending the concrete domains with more binary predicates, e.g. multiplication, while preserving first order rewritability of CQs. On the other hand, we would like to study query rewriting w.r.t. with other concrete domains, for instance those over strings. We believe that the rules we have introduced in this paper could serve as a basis for a more generalised procedure supporting those domains.

## References

[1] Christian Alrabbaa, Patrick Koopmann, and Anni-Yasmin Turhan. Practical Query Rewriting for *DL-Lite* with Numerical Predicates (extended version). LTCS-Report 19-06, Chair for Automata

Theory, Institute for Theoretical Computer Science, Technische Universität Dresden, Dresden, Germany, 2019. See https://lat.inf.tu-dresden.de/research/reports.html.

[2] Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The *DL-Lite* Family and Relations. *Journal of Artificial Intelligence Research*, 2009.

[3] Alessandro Artale, Vladislav Ryzhikov, and Roman Kontchakov. *DL-Lite* with Attributes and Datatypes. In *ECAI 2012: 20th European Conference on Artificial Intelligence (ECAI'12)*, 2012.

[4] Franz Baader, Stefan Borgwardt, and Marcel Lippmann. Query rewriting for *DL-Lite* with $n$-ary concrete domains. In *Proc. of the 26th International Joint Conference on AI (IJCAI'17)*, 2017.

[5] Franz Baader and Philipp Hanschke. A Scheme for Integrating Concrete Domains into Concept Languages. In *Proc. of the 12th International Joint Conference on AI (IJCAI'91)*, 1991.

[6] Franz Baader, Patrick Koopmann, and Anni-Yasmin Turhan. Using ontologies to query probabilistic numerical data. In *Proc. of Frontiers of Combining Systems, (FroCoS'17)*, 2017.

[7] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family. *Journal of Automated Reasoning*, 2007.

[8] David Carral, Irina Dragoste, and Markus Krötzsch. The Combined Approach to Query Answering in *Horn-ALCHOIQ*. In *Proceedings of KR'18*, 2018.

[9] Roman Kontchakov, Carsten Lutz, David Toman, Frank Wolter, and Michael Zakharyaschev. The Combined Approach to Ontology-Based Data Access. In *Proc. of the 22nd International Joint Conference on Artificial Intelligence,(IJCAI'11)*, 2011.

[10] Carsten Lutz. Inverse roles make conjunctive queries hard. In *Proc. of the 20th International Workshop on Description Logics, (DL'07)*, 2007.

[11] Carsten Lutz, David Toman, and Frank Wolter. Conjunctive Query answering in the Description Logic $\mathcal{EL}$ Using a Relational Database System. In *Proc. of the 21st International Joint Conference on Artificial Intelligence, (IJCAI'09)*, 2009.

[12] Nhung Ngo, Magdalena Ortiz, and Mantas Simkus. Closed Predicates in Description Logics: Results on Combined Complexity. In *Proceedings of KR'16*, 2016.

[13] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *Journal on Data Semantics*, 2008.

[14] Sebastian Rudolph and Birte Glimm. Nominals, inverses, counting, and conjunctive queries or: why infinity is your friend! *Journal of Artificial Intelligence Research*, 2010.

[15] Ognjen Savkovic and Diego Calvanese. Introducing Datatypes in *DL-Lite*. In *ECAI 2012 - 20th European Conference on Artificial Intelligence (ECAI'12)*, 2012.