A Pragmatic Approach to Answering CQs over Fuzzy *DL-Lite*-ontologies–introducing FLite ⋆

Theofilos Mailis[1], Anni-Yasmin Turhan[2], and Erik Zenker[3]

[1] Department of Informatics and Telecommunications,
National and Kapodistrian University of Athens, Greece
[2] Department of Computer Science, University of Oxford, UK
[3] Institute for Theoretical Computer Science,
Technische Universität Dresden, Germany

Abstract. Fuzzy Description Logics (FDLs) generalize crisp ones by providing membership degree semantics. To offer efficient query answering for FDLs it is desirable to extend the rewriting-based approach for *DL-Lite* to its fuzzy variants. For answering conjunctive queries over fuzzy *DL-Lite*$_R$ ontologies we present an approach, that employs the crisp rewriting as a black-box procedure and treats the degrees in a second rewriting step. This pragmatic approach yields a sound procedure for the Gödel based fuzzy semantics, which we have implemented in the FLite reasoner that employs the Ontop system. A first evaluation of FLite suggests that one pays only a linear overhead for fuzzy queries.

1 Introduction

Fuzzy variants of Description Logics (DLs) were introduced in order to describe concepts for which there exists no sharp, unambiguous distinction between members and nonmembers. For example, a natural way to model a component running at half of its capacity is to state that is overused to a degree of, say, 0.6.

In the last years, conjunctive query answering has been investigated for crisp and fuzzy DLs. The *DL-Lite* family of DLs [2, 6, 5] guarantees that query answering can be done efficiently —in the size of the data and in the overall size of the corresponding ontology. Though fuzzy *DL-Lite* variants have already been successfully investigated [18, 19, 10], their algorithms do not exploit the optimizations of query rewriting techniques that have been implemented in many systems for the crisp DLs, such as QuOnto2 [1, 12], Ontop [14], Owlgres [16], and IQAROS [22].

A pragmatic approach for answering conjunctive queries over crisp *DL-Lite*$_R$-TBoxes and fuzzy *DL-Lite*$_R$-ABoxes, that takes advantage of these optimizations, was presented in [9]. The combination of crisp TBoxes with fuzzy ABoxes is useful in applications, where only the data is imprecise, while the terminological knowledge is not, e.g., as in situation recognition applications that rely on sensor data.

⋆ Partially supported by DFG in the Collaborative Research Center 912 "Highly Adaptive Energy-Efficient Computing".

In our pragmatic approach to answering of (fuzzy) conjunctive queries, a crisp *DL-Lite* reasoner is used as a black box to obtain an initial rewriting of the conjunctive query (without the fuzzy degrees). The obtained query gets extended in a second rewriting step by (1) fuzzy atoms, (2) degree variables that capture numerical membership degrees, and (3) numerical predicates that realize the fuzzy operators. The resulting query can then be evaluated by a SQL engine. This simple approach yields a sound and complete implementation only if fuzzy semantics with the min operator for conjunction is employed as, for instance, the popular Gödel semantics. For other semantics, at it will later be explained, our approach remains complete but not sound. We have implemented this approach in the FLite prototype, which uses the optimized Ontop reasoner to generate the first rewriting.

In this paper we describe the pragmatic approach, introduce some optimizations of it and study the scalability of FLite—in particular the overhead introduced by reasoning over fuzzy information. To this end we evaluate the FLite implementation against Ontop for different (fuzzy) conjunctive queries performed on a TBox and fuzzy ABoxes of different sizes. The benchmarks used for the evaluation are based on a situation recognition application for complex hard- and software systems.

The rest of the paper is structured as follows: Section 2 recalls *DL-Lite*$_R$ and its fuzzy variant. Section 3 presents a description of the two-step rewriting approach for answering fuzzy conjunctive queries and we discuss its limitations. In Section 4 we examine optimizations for this approach, which are implemented in FLite. Section 5 introduces our application of situation recognition that motivates the two-step rewriting approach for querying crisp TBoxes with fuzzy ABoxes and a benchmark from this application. The detailed evaluation of the FLite system against Ontop is given in Section 6 for different ontology sizes and different fuzzy queries. Conclusions and future work end the paper in Section 7.

2 Preliminaries

We start with *DL-Lite*$_R$ [2] and introduce its fuzzy variant [19, 10] afterwards. Let the following be countably infinite and pairwise disjoint sets: $\mathsf{N_C}$ of concept names, $\mathsf{N_R}$ of role names, $\mathsf{N_I}$ of individual names, and $\mathsf{N_V}$ of variable names. From these sets the complex *DL-Lite*$_R$-concepts, -roles and -queries are constructed. *DL-Lite*$_R$-concepts and -roles are defined according to the following grammar:

$$B \to A \mid \exists Q \qquad C \to \top \mid B \mid \neg B \qquad Q \to P \mid P^- \qquad R \to Q \mid \neg Q,$$

where $\top$ is the top concept, $A \in \mathsf{N_C}$, $P \in \mathsf{N_R}$. Based on these kinds of complex concepts and roles, a *DL-Lite*$_R$ TBox $\mathcal{T}$ is a finite set of axioms of the form: $B \sqsubseteq C$, $Q \sqsubseteq R$ or $funct(Q)$. Let $a, b \in \mathsf{N_I}$ and $d \in [0, 1]$ a *fuzzy degree*. A *fuzzy assertion* is of the forms: $\langle B(a), \geqslant d\rangle$ or $\langle P(a, b), \geqslant d\rangle$. An *ABox* $\mathcal{A}$ is a finite set of fuzzy assertions. A *fuzzy DL-Lite-ontology* $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ consists of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. Please note that the TBoxes are always crisp in our setting.

Table 1. Families of fuzzy logic operators.

Family	t-norm $a \otimes b$	negation $\ominus a$	implication $\alpha \Rightarrow b$
Gödel	$\min(a, b)$	$\begin{cases} 1, & a = 0 \\ 0, & a > 0 \end{cases}$	$\begin{cases} 1, & a \leqslant b \\ b, & a > b \end{cases}$
Łukasiewicz	$\max(a + b - 1, 0)$	$1 - a$	$\min(1 - a + b, 1)$
Product	$a \times b$	$\begin{cases} 1, & a = 0 \\ 0, & a > 0 \end{cases}$	$\begin{cases} 1, & a \leqslant b \\ b/a, & a > b \end{cases}$

Crisp *DL-Lite*$_R$-ontologies are a special case of fuzzy ones, where only degrees 1 and 0 are admitted.

The reasoning problem we address is answering of (unions of) conjunctive queries. Let $t_1, t_2 \in \mathsf{N_I} \cup \mathsf{N_V}$ be terms, an *atom* is an expression of the form: $C(t_1)$ or $P(t_1, t_2)$. Let $\mathbf{x}$ and $\mathbf{y}$ be vectors over $\mathsf{N_V}$, then $\phi(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms of the forms $A(t_1)$ and $P(t_1, t_2)$. A *conjunctive query* (CQ) $q(\mathbf{x})$ over an ontology $\mathcal{O}$ is a first-order formula $\exists \mathbf{y}.\phi(\mathbf{x}, \mathbf{y})$, where $\mathbf{x}$ are the *answer variables*, $\mathbf{y}$ are *existentially quantified variables* and the concepts and roles in $\phi(\mathbf{x}, \mathbf{y})$ appear in $\mathcal{O}$. Observe, that the atoms in a CQ do not contain degrees. A *union of conjunctive queries* (UCQ) is a finite set of conjunctive queries that have the same number of answer variables.

The *semantics* of fuzzy *DL-Lite*$_R$ is provided via the different families of fuzzy logic operators depicted in Table 1 and interpretations. An *interpretation* for fuzzy *DL-Lite*$_R$ is a pair $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta^\mathcal{I}$ is as usual, but $\cdot^\mathcal{I}$ is an interpretation function mapping every

- $a \in \mathsf{N_I}$ to some element $a^\mathcal{I} \in \Delta^\mathcal{I}$,
- $A \in \mathsf{N_C}$ to a *concept membership function* $A^\mathcal{I} : \Delta^\mathcal{I} \to [0, 1]$,
- $P \in \mathsf{N_R}$ to a *role membership function* $P^\mathcal{I} : \Delta^\mathcal{I} \times \Delta^\mathcal{I} \to [0, 1]$.

Let δ, δ' denote elements of $\Delta^\mathcal{I}$ and $\ominus$ denote fuzzy negation (Table 1), then the semantics of concepts and roles are inductively defined as follows:

$$(\exists Q)^\mathcal{I}(\delta) = \sup_{\delta' \in \Delta^\mathcal{I}} Q^\mathcal{I}(\delta, \delta') \qquad (\neg B)^\mathcal{I}(\delta) = \ominus B^\mathcal{I}(\delta) \qquad \top^\mathcal{I}(\delta) = 1$$

$$P^{-\mathcal{I}}(\delta, \delta') = P^\mathcal{I}(\delta', \delta) \qquad (\neg Q)^\mathcal{I}(\delta, \delta') = \ominus Q^\mathcal{I}(\delta, \delta')$$

An interpretation $\mathcal{I}$ *satisfies* $B \sqsubseteq C$ iff $B^\mathcal{I}(\delta) \leqslant C^\mathcal{I}(\delta)$ for every $\delta \in \Delta^\mathcal{I}$, $Q \sqsubseteq R$ iff $Q^\mathcal{I}(\delta, \delta') \leqslant R^\mathcal{I}(\delta, \delta')$ for every $\delta, \delta' \in \Delta^\mathcal{I}$, and $\mathrm{func}(Q)$ iff for every $\delta \in \Delta^\mathcal{I}$ there is a unique $\delta' \in \Delta^\mathcal{I}$ such that $Q^\mathcal{I}(\delta, \delta') > 0$. An interpretation $\mathcal{I}$ is a *model of a TBox* $\mathcal{T}$, i.e. $\mathcal{I} \models \mathcal{T}$, iff it satisfies all axioms in $\mathcal{T}$. $\mathcal{I}$ satisfies $\langle B(a), \geqslant d \rangle$ iff $B^\mathcal{I}(a^\mathcal{I}) \geqslant d$, and $\langle P(a, b), \geqslant d \rangle$ iff $P^\mathcal{I}(a^\mathcal{I}, b^\mathcal{I}) \geqslant d$. $\mathcal{I}$ is a *model of an ABox* $\mathcal{A}$, i.e. $\mathcal{I} \models \mathcal{A}$, iff it satisfies all assertions in $\mathcal{A}$. Finally an interpretation $\mathcal{I}$ is a model of an ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ iff it is a model of $\mathcal{A}$ and $\mathcal{T}$.

Given a CQ $q(\mathbf{x}) = \exists \mathbf{y}.\phi(\mathbf{x}, \mathbf{y})$, an interpretation $\mathcal{I}$, a vector of individuals $\boldsymbol{\alpha}$ with the same arity as $\mathbf{x}$, we define the mapping π that maps: i) each individual

a to $a^{\mathcal{I}}$, ii) each variable in $\mathbf{x}$ to an element of $\boldsymbol{\alpha}^{\mathcal{I}}$, and iii) each variable in $\mathbf{y}$ to an element $\delta \in \Delta^{\mathcal{I}}$. Suppose that for an interpretation $\mathcal{I}$, Π is the *set of mappings* that comply to these three conditions. Computing the t-norm $\otimes$ of all atoms: $A^{\mathcal{I}}(\pi(t_1))$ and $P^{\mathcal{I}}(\pi(t_1), \pi(t_2))$ yields the degree of $\phi^{\mathcal{I}}(\boldsymbol{\alpha}^{\mathcal{I}}, \pi(\mathbf{y}))$. A tuple of individuals $\boldsymbol{\alpha}$ is a *certain answer* to $q(\mathbf{x})$, over $\mathcal{O}$, with a degree greater or equal than d (denoted $\mathcal{O} \models q(\boldsymbol{\alpha}) \geqslant d$), if for every model $\mathcal{I}$ of $\mathcal{O}$:

$$q^{\mathcal{I}}(\boldsymbol{\alpha}^{\mathcal{I}}) = \sup_{\pi \in \Pi}\{\phi^{\mathcal{I}}(\boldsymbol{\alpha}, \pi(\mathbf{y}))\} \geqslant d.$$

We denote the set of certain answers along with degrees, to a query $q(\mathbf{x})$ w.r.t. an ontology $\mathcal{O}$ with $ans(q(\mathbf{x}), \mathcal{O})$:

$$ans(q(\mathbf{x}), \mathcal{O}) = \{(\boldsymbol{\alpha}, d) \mid \mathcal{O} \models q(\boldsymbol{\alpha}) \geqslant d \wedge \nexists d'.d' > d \wedge \mathcal{O} \models q(\boldsymbol{\alpha}) \geqslant d'\}.$$

To illustrate the use of the fuzzy *DL-Lite*$_R$ language and queries, we provide an example from our application domain.

Example 1. The ontology $\mathcal{O}_{ex}$ for our running example consists of:

$$\begin{aligned}
\mathcal{T}_{ex} &:= \{\text{Server} \sqsubseteq \exists \text{hasCPU},\ \exists \text{hasCPU}^- \sqsubseteq \text{CPU},\ \text{func}(\text{hasCPU}^-)\} \\
\mathcal{A}_{ex} &:= \{\langle \text{Server}(\text{server}_1), \geqslant 1\rangle,\ \langle \text{hasCPU}(\text{server}_1, \text{cpu}_1), \geqslant 1\rangle, \\
&\quad \langle \text{OverUsed}(\text{cpu}_1), \geqslant 0.6\rangle,\ \langle \text{hasCPU}(\text{server}_1, \text{cpu}_2), \geqslant 1\rangle, \\
&\quad \langle \text{OverUsed}(\text{cpu}_2), \geqslant 0.8\rangle \qquad\qquad \}
\end{aligned}$$

The first two axioms in $\mathcal{T}_{ex}$ state that each server has a part that is a CPU. The third one states that no CPU can belong to more than one server. $\mathcal{A}_{ex}$ provides information about the connections between servers and CPUs and each CPU's degree of overuse. To query the ontology $\mathcal{O}_{ex}$ we can formulate the queries:

$$\begin{aligned}
q_1(x, y) &= \text{hasCPU}(x, y) \wedge \text{OverUsed}(y) \\
q_2(x) &= \exists y\ \text{hasCPU}(x, y) \wedge \text{OverUsed}(y)
\end{aligned}$$

The query q_1 asks for pairs of Servers and CPUs with an overused CPU. The query q_2 asks for Servers, where the Server's CPU is overused. If conjunction and negation are interpreted as the Gödel family of operators, the certain answers w.r.t. $\mathcal{O}_{ex}$ are:

$$\begin{aligned}
ans(q_1(x, y), \mathcal{O}_{ex}) &= \{(\text{server}_1, \text{cpu}_1, 0.6),\ (\text{server}_2, \text{cpu}_2, 0.8)\} \\
ans(q_2(x), \mathcal{O}_{ex}) &= \{(\text{server}_1, 0.8)\}.
\end{aligned}$$

3 Fuzzy Query Answering by Extended Crisp Rewritings

Let CQ $q(\mathbf{x})$ be formulated over the vocabulary of the *DL-Lite*$_R$ ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$. The main idea underlying the classic *DL-Lite*$_R$ query answering algorithm is to rewrite the query $q(\mathbf{x})$ with the information from the TBox $\mathcal{T}$ into a UCQ $q_{\mathcal{T}}(\mathbf{x})$ and then apply this UCQ to the ABox $\mathcal{A}$ alone [6, 2]. For fuzzy

DLs we extend this approach to handle degrees of ABox assertions. The main idea is depicted in Figure 1. To explain the algorithm we need the predicates A_f, P_f, and $\Phi_\otimes$. Intuitively, each binary predicate A_f is an extension of the unary predicate A such that the fuzzy concept assertion $\langle A(a), \geqslant d\rangle$ is equivalent to the predicate assertion $A_f(a, d)$ (similarly for P_f). The n-ary predicate $\Phi_\otimes$ is adopted in order to realize the semantics of the fuzzy conjunction (e.g. minimum for the Gödel norm) within a FOL query. Thus, for each tuple of degrees $d_1, \ldots, d_n \in [0, 1]$ such that $d_1 = \otimes(d_2, \ldots, d_n)$, we have that $(d_1, \ldots, d_n) \in \Phi_\otimes$. Suppose now that the CQ $q(\mathbf{x})$ is to be answered. The two-step rewriting algorithm proceeds as follows:

1. The crisp *DL-Lite*$_R$ algorithm rewrites $q(\mathbf{x})$ to $q_\mathcal{T}(\mathbf{x})$. (For ease of presentation we assume that $q_\mathcal{T}(\mathbf{x})$ is still a CQ.)
2. The fuzzy query $q_{\mathcal{T},f}(\mathbf{x}, x_d)$ is computed from $q_\mathcal{T}(\mathbf{x})$ by replacing atoms of the form $A(t_1)$ and $P(t_1, t_2)$ by $A_f(t_1, y_d)$ and $P_f(t_1, t_2, y_d)$, where the variable y_d is a *degree variable*. Its purpose is to retrieve the degree of an assertion. The degree value for fuzzy conjunction is retrieved by the predicate $\Phi_\otimes$ and (to be) stored in the additional degree variable x_d. Thus, the conjunction degree of a new atom $q_{\mathcal{T},f}(\mathbf{x}, x_d)$ is obtained by the predicate $\Phi_\otimes(x_d, y_1, \ldots, y_n)$, where y_i is a degree variable in the ith atom of the CQ.
3. The query is evaluated over the ABox and the actual computation of the degree values takes place. Now, for a tuple of individuals $\boldsymbol{\alpha}$ and degrees $d_1, d_2 \in [0, 1]$, if (α, d_1) and (α, d_2) are both answers to the query, only the answer with the higher degree is returned.

Note that this description abstracts from the fact that the ABox $\mathcal{A}$ is usually implemented by a relational database $\mathcal{D}$ and a mapping $\mathcal{M}$. We see in Section 3.1 how this mapping is extended to incorporate fuzzy information. For a more detailed presentation of the algorithms, the reader may refer to [9].

Example 2. We return to Example 1 and illustrate the application of the algorithm to the queries. Initially, q_1 and q_2 are rewritten to the following UCQs:

$$q_{1\mathcal{T}_{ex}}(x, y) = \{\text{hasCPU}(x, y) \wedge \text{OverUsed}(y)\}$$
$$q_{2\mathcal{T}_{ex}}(x) = \{\exists y.\text{hasCPU}(x, y) \wedge \text{OverUsed}(y)\}$$

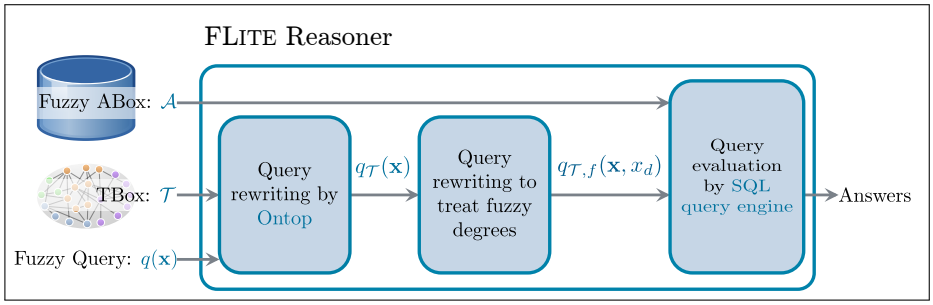

Fig. 1. The process flow diagram of the FLITE rewriting procedure.

In the next step, the algorithm extends the queries with degree variables and atoms, so that the corresponding degrees can be returned:

$$q_1{}^{f}_{\mathcal{T}_{ex}}(x, y, x_d) = \{\text{hasCPU}(x, y, y_{d_1}) \wedge \text{OverUsed}(y, y_{d_2}) \wedge \Phi_{\otimes}(x_d, y_{d_1}, y_{d_2})\}$$
$$q_2{}^{f}_{\mathcal{T}_{ex}}(x, x_d) = \{\exists y.\text{hasCPU}(x, y, y_{d_1}) \wedge \text{OverUsed}(y, y_{d_2}) \wedge \Phi_{\otimes}(x_d, y_{d_1}, y_{d_2})\}$$

For the ABox $\mathcal{A}_{ex}$ the following set of answers to each of the queries are returned:

$$ans(q_1{}^{f}_{\mathcal{T}_{ex}}(x, x_d), \mathcal{A}_{ex}) = \{(server_1, cpu_1, 0.6), (server_1, cpu_2, 0.8)\}$$
$$ans(q_2{}^{f}_{\mathcal{T}_{ex}}(x, x_d), \mathcal{A}_{ex}) = \{(server_1, 0.8)\}.$$

The limitations of our pragmatic approach are explained in [9]. To sum up, our approach yields sound and complete results for fuzzy semantics based on the min t-norm operator such as the Gödel family of operators. The correctness for this case can be derived from the crisp *DL-Lite*$_R$ proof along with the following points: (1) only crisp TBox axioms are allowed, (2) conjunctions only appear in conjunctive query expressions, (3) Ontop optimizations do not affect the correctness of the algorithm due to the properties of the min operator. To illustrate the last point a conjunctive query $q_{\mathcal{T}}(x) := A(x) \wedge A(x)$ is simplified by Ontop to $q_{\mathcal{T}}(x) := A(x)$. This simplification is correct for the min operator since $\min(A^{\mathcal{I}}(\delta), A^{\mathcal{I}}(\delta)) = A^{\mathcal{I}}(\delta)$ (for every interpretation $\mathcal{I}$ and every $\delta \in \Delta^{\mathcal{I}}$) . The latter does not apply for other t-norms, therefore the proposed methodology is complete but not sound for the Łukasiewicz and Gödel families of operators. Nevertheless, as described in [9], we devised a method by which each unsound answer can be identified and its correct degree estimated between two membership values. I.e. our algorithm asserts that for some $d \in [0.7, 0.8]$, $(server_1, d) \in ans(q_3{}^{f}_{\mathcal{T}_{ex}}(x, x_d), \mathcal{A}_{ex})$.

3.1 The FLite Reasoner Implementation

FLite[4] (Fuzzy *DL-Lite*$_R$ query engine) implements the presented query answering algorithm and builds on the Ontop framework [14]. On a technical level, the rewriting procedure becomes more involved when a reasoner such as Ontop is deployed, since such systems are build to operate on relational databases. Thus the queries $q_{\mathcal{T}}(\mathbf{x})$ and $q_{\mathcal{T},f}(\mathbf{x}, x_d)$ in Figure 1 are SQL queries, while the ABox $\mathcal{A}$ is derived from a partial mapping:

$$\mathcal{M} : \text{SQL SELECT Statements} \rightarrow \text{ABox assertions.}$$

In order to embed fuzzy information into mappings we adopt a reification approach sketched in the following example.

Example 3. We consider a fuzzy mapping described in the Quest syntax [13]. In this mapping, for the concept popularVideo each id of the table videos is

[4] The FLite reasoner is available from the following Git: https://iccl-share.inf.tu-dresden.de/flite-developer/flite.git

annotated with a popularity, i.e. a fuzzy degree:

```
target haec:video_{popularity_degree} a haec:PopularVideo.
source SELECT   fuzzy(id, popularity)
       AS       popularity_degree
       FROM     videos
```

Now, if a video with an id of 12 and a popularity of 0.8 appears in the database, then this corresponds to $\langle \text{PopularVideo}(\mathit{videos}12), \geqslant 0.8 \rangle$ stated in the ontology.

The function *fuzzy(column, degree)* is a marker for the FLite parser to recognize fuzzy statements. It indicates that every element of the particular *column* (or SQL expression) gets a fuzzy *degree* assigned. This degree can either be a column with values in $[0, 1]$, or an SQL expression corresponding to a fuzzy membership function. It should be noted that SQL expressions containing the fuzzy marker function appear in the initial mapping $\mathcal{M}$ and in $q_{\mathcal{T}}(\mathbf{x})$ queries while these markers are converted in $q_{\mathcal{T},f}(\mathbf{x}, x_d)$ queries to a SQL expressions that return the membership degree.

4 Optimizations for the FLite Implementation

Implementing the pragmatic approach naively would be very inefficient. First, in contrast to Ontop's rewritings, the "fuzzy" SQL query resulting from the FLite rewriting process is not optimized for fast execution. Thus, the query engine retrieves the same items multiple times. Second, the database contains numerical values, that are mapped to coarser categories with membership degrees at query execution time. These overheads can be reduced by optimizing the fuzzy rewritings and fuzzifying numerical values in the database on a preprocessing step. These optimizations are discussed in the following.

Self-join Removal (SR). Ontop performs a restructuring of query rewritings as outlined in [15, 7]. Due to the reification process for embedding fuzzy information, some of the optimizations performed by Ontop are obscured by the *fuzzy* pseudo-function. Currently, Ontop appears to omit the optimization step, if an unknown function is found in the query. Thus, its optimizer is not aware of the fact that the *fuzzy* function is exclusively used by FLite to tag fuzzy degrees in the query. Extending the process shown in Figure 1, the query optimizer should be applied to the query $q_{\mathcal{T},f}(\mathbf{x}, x_d)$, e.g. to remove self-joins. To illustrate the problem, consider a query $q_{\mathcal{T},f}$ that contains self-joins of the following form:

```
SELECT a.column_a, a.degree_a, b.column_b
FROM table_A as a,
INNER JOIN table_A as b ON a.column_a=b.column_a
```

It can easily be verified that this query contains redundant selection statements over the same database table and can be simplified to:

```
SELECT column_a, degree_a, column_b FROM table_A
```

The second query is performed in linear time w.r.t. the size of table_A, while the first one is performed in quadratic time. Indeed, we observed a huge improvement on query execution time, depending on the number of tables that are self-joined.

Pre-computation of Membership Degrees (PD). Calculation of membership degrees of numerical values during query execution can lead to significant run-time overhead depending on the complexity of the applied membership function. The following listing shows an SQL statement where fuzzy degrees are assigned to the values of a column by the membership degree function f:

```
SELECT fuzzy(column_a, f(column_b)) FROM crisp_table
```

The case where membership degrees are available directly , reduces the overhead of calling and executing such a function. In the following SQL statement, function f was replaced by a membership degree column.

```
SELECT fuzzy(column_a, membership_degree_column)
FROM fuzzy_table
```

For simple mappings of this form, database schema restructuring is unnecessary, since the observed raw data in our application database are often numerical values, an automatic mechanism that translates these values to membership degrees is necessary. For this purpose we introduce computed or virtual columns, which contain values that are computed by a user-defined membership function, which is triggered when a new row is added to the table. This avoids the overhead for the computation of membership function during the evaluation of the rewritten query at the cost of an increase in database size and membership function overhead during the database update process. In the end he FLite user has to assess on the basis of the mapping whether to spend more time for query execution or more disc space for additional fuzzy columns.

5 A Sample Application: Situation Recognition

The project "Highly Adaptive Energy-efficient Computing" (HAEC) investigates complex computing environments that are highly energy-efficient while compromising utility of services as little as possible. In order to be adaptive, the system needs to trigger adaptations (of hard- or software components), if the quality of the requested services or their number changes. To provide such a trigger mechanism we investigate an ontology-based situation recognition. The situations to be recognized are modelled as conjunctive queries. The background information on the system is captured in the TBox and the current system's state is captured by an ABox. Such ABoxes are automatically generated from sensor data and other systems information and the conjunctive queries for the situations are evaluated. In such a setting the numerical sensor data need to be mapped to coarser, symbolic categories and membership degrees. Similarly, the query needs to be able to retrieve individuals that fulfill the conditions of the query to a degree. We have built a TBox and a collection of ABoxes and queries for this application.

5.1 Towards a Benchmark

The hard- and software components of the HAEC system are modeled in a TBox which consists of 197 GCIs, 168 named classes and 38 roles (415 axioms total). Each state of the HAEC system is stored in tables of a relational database. This database stores the information on the soft- and hardware of the HAEC system. The tables contain numerical values for boards, processes, requests, etc.[5] Identifying the HAEC ontology as TBox and the HAEC database as ABox, this benchmark compares the run-time to answer (fuzzy) conjunctive queries over TBox and ABox by Ontop and FLite. In contrast to Ontop which requires a crisp mapping, FLite is using a partially fuzzy mapping.

6 Results of FLite on the Benchmark

FLite was evaluated over a series of variations of the HAEC benchmarks. The number of additional atoms in the conjunctive query n and the database scale factor k were increased in the series.

The initial database with a size of 768.0 KiB was scaled by k in the range of 10^0 to 10^5 to about 13 GiB by the Virtual Instance Generator (VIG) [8]. For each of these scaled databases, the number of atoms of the initial conjunctive query (2 concepts, 4 roles) was increased by n additional atoms from one atom to a maximum of seven (13 atoms total). The system on which the benchmark was performed on is powered by an Intel Core i7 2.6 GHz processor and was equipped with 8 GB DDR 1600 main memory. ABox information was stored in a MySQL 5.6.23 database. FLite and Ontop are executed on Oracle the JVM 1.7.

Figure 2 shows the query execution time of Ontop for a query with seven additional atoms and with a crisp mapping compared to the naive FLite implementation for a query with one to seven additional atoms with fuzzy mapping using Gödel t-norms for an increasing database size.
FLite needs longer run-time in the order of almost two magnitudes compared to the one of Ontop. Therefore, it is vital that the naive FLite implementation is enhanced by optimization techniques. Employing the optimizations introduced in Section 4 results in the query execution times displayed in Figure 3.

Optimization SR in Figure 3 shows run-time reduction up to a scale factor of about thousand, then it converges with the naive FLite implementation. Thus, an optimization that also effect larger databases is necessary. Optimization PD in Figure 3 shows opposite behavior, resulting in a run-time reduction from a scale factor of thousand. Finally, a combination of both reduces the run-time on all scale factors and is even on the same level with Ontop up to a scale factor of thousand. Thus, a query execution time with a flat linear run-time overhead with respect to Ontop is possible when the fuzzy rewriting is optimized and fuzzy translation functions are replaced by fuzzy computed columns.

[5] The files necessary to perform this benchmark can be found here: https://iccl-share.inf.tu-dresden.de/erikzenker/flite-benchmark.git

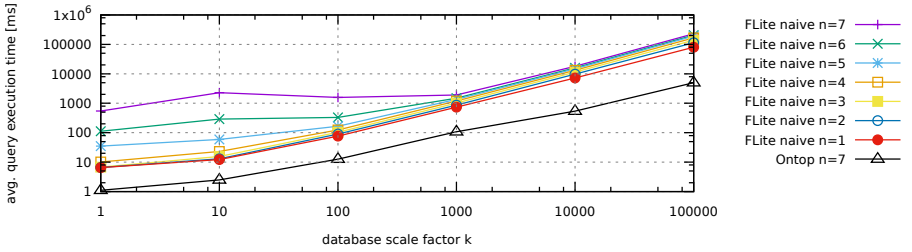

Fig. 2. Query execution time of ONTOP with seven additional atoms with crisp mapping compared to the naive FLITE implementation with one to seven additional atoms with fuzzy mapping.

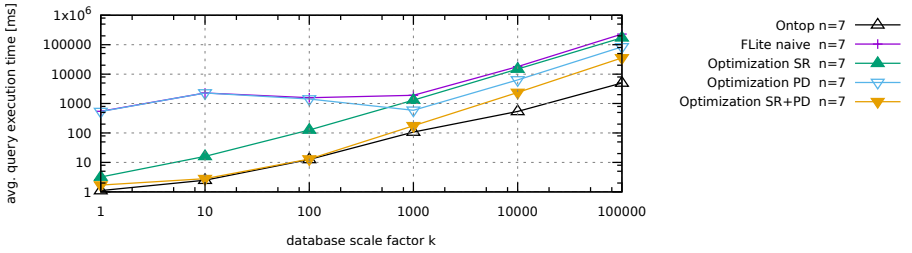

Fig. 3. ONTOP is compared to optimized FLITE versions SR, PD, and SR+PD. All setups were executed over a CQ with seven additional atoms.

Instead of comparing FLITE with ONTOP, a comparison with other fuzzy DL reasoners would have been desirable. However, reasoners such as LiFR [21], FuzzyDL [4], FiRE [17], and DeLorean [3] support only instance checking rather than conjunctive query answering (albeit for more expressive DLs than $DL\text{-}Lite_R$). Others such as the *DL-Lite* reasoner Ontosearch2 [11] or SoftFacts [20] could either not be obtained or installed. Thus we had to resort to ONTOP for a comparison for query answering in $DL\text{-}Lite_R$.

7 Conclusions

We presented a pragmatic approach for answering fuzzy conjunctive queries over $DL\text{-}Lite_R$-ontologies with fuzzy ABoxes. Our approach uses rewritings obtained by the algorithm for answering crisp queries as an intermediate step and thus allows to make use of standard query rewriting engines. Although described here for $DL\text{-}Lite_R$, our approach can be extended to other DLs that enjoy FOL rewritability. Our algorithm is sound for those t-norms that have idempotent operators, such as the Gödel t-norm.

We implemented our approach in the FLITE system and evaluated it against the ONTOP reasoner for ABoxes of varying size. Our evaluation gave evidence that there is a substantial increase of run-time for large ABoxes, when fuzzy

information is queried. This increase can be reduced by basic optimizations. However, developing and extending FLite is on-going work.

References

1. A. Acciarri, D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, and R. Rosati. QUONTO: Querying Ontologies. In *AAAI*, pages 1670–1671, 2005.
2. A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev. The *DL-Lite* Family and Relations. *Journal of artificial intelligence research*, 36(1):1–69, 2009.
3. F. Bobillo, M. Delgado, and J. Gómez-Romero. Reasoning in Fuzzy OWL 2 with DeLorean. In *Uncertainty Reasoning for the Semantic Web II*, pages 119–138. 2013.
4. F. Bobillo and U. Straccia. fuzzyDL: An Expressive Fuzzy Description logic Reasoner. In *FUZZ-IEEE*, pages 923–930, 2008.
5. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, and R. Rosati. *Ontologies and Databases: The DL-Lite Approach*. 2009.
6. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family. *Journal of Automated reasoning*, 39, 2007.
7. R. Kontchakov, M. Rezk, M. Rodríguez-Muro, G. Xiao, and M. Zakharyaschev. Answering SPARQL Queries over Databases under OWL 2 QL Entailment Regime. In *The Semantic Web–ISWC 2014*, pages 552–567. Springer, 2014.
8. D. Lanti, M. Rezk, M. Slusnys, G. Xiao, and D. Calvanese. The NPD Benchmark for OBDA Systems. In *10th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2014)*, page 3, 2014.
9. T. Mailis and A.-Y. Turhan. Employing DL-Lite$_R$-Reasoners for Fuzzy Query Answering. In *Proceedings of the 4th Joint International Semantic Technology Conference (JIST2014)*, LNCS, 2014.
10. J. Z. Pan, G. B. Stamou, G. Stoilos, and E. Thomas. Expressive Querying over Fuzzy DL-Lite Ontologies. In *Description Logics*, 2007.
11. J. Z. Pan, E. Thomas, and D. Sleeman. Ontosearch2: Searching and querying web ontologies. *Proc. of WWW/Internet*, 2006:211–218, 2006.
12. A. Poggi, M. Rodriguez, and M. Ruzzi. Ontology-based database access with DIG-Mastro and the OBDA Plugin for Protégé. In *Proc. of OWLED*, 2008.
13. M. Rodriguez-Muro, J. Hardi, and D. Calvanese. Quest: Efficient SPARQL-to-SQL for RDF and OWL. In *11th International Semantic Web Conference ISWC 2012*, page 53, 2012.
14. M. Rodriguez-Muro, R. Kontchakov, and M. Zakharyaschev. Ontology-Based Data Access: Ontop of Databases. In *International Semantic Web Conference (1)*, volume 8218 of *LNCS*, pages 558–573. Springer, 2013.
15. M. Rodriguez-Muro, M. Rezk, J. Hardi, M. Slusnys, T. Bagosi, and D. Calvanese. Evaluating SPARQL-to-SQL Translation in Ontop. In *Informal Proceedings of ORE'13*, pages 94–100, 2013.
16. M. Stocker and M. Smith. Owlgres: A Scalable OWL Reasoner. In *OWLED*, volume 432, 2008.
17. G. Stoilos, N. Simou, G. Stamou, and S. Kollias. Uncertainty and the Semantic Web. *Intelligent Systems*, 21(5):84–87, 2006.
18. U. Straccia. Answering Vague Queries in Fuzzy DL-Lite. In *Proceedings of the 11th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems,(IPMU-06)*, pages 2238–2245, 2006.

19. U. Straccia. Towards Top-k Query Answering in Description Logics: The Case of DL-Lite. In *Logics in Artificial Intelligence*, pages 439–451. Springer, 2006.
20. U. Straccia. SoftFacts: A Top-k Retrieval Engine for Ontology Mediated Access to Relational Databases. In *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*, pages 4115–4122. IEEE, 2010.
21. D. Tsatsou, S. Dasiopoulou, I. Kompatsiaris, and V. Mezaris. LiFR: A Lightweight Fuzzy DL Reasoner. In *The Semantic Web: ESWC 2014 Satellite Events*, pages 263–267. 2014.
22. T. Venetis, G. Stoilos, and G. Stamou. Query Extensions and Incremental Query Rewriting for OWL 2 QL Ontologies. *Journal on Data Semantics*, pages 1–23, 2014.