Towards Temporal Fuzzy Query Answering on Stream-based Data[⋆]

Anni-Yasmin Turhan[1] and Erik Zenker[2]

[1] Department of Computer Science, University of Oxford, UK
[2] Institute for Theoretical Computer Science,
Technische Universität Dresden, Germany

Abstract. For reasoning over streams of data ontology-based data access is a common approach. The method for answering conjunctive queries (CQs) over *DL-Lite* ontologies in this setting is by rewritings of the query and evaluation of the resulting query by a data base engine. For stream-based applications the classical expressivity of *DL-Lite* lacks means to handle fuzzy and temporal information. In this paper we report on a combination of a recently proposed pragmatic approach for answering CQs over fuzzy *DL-Lite* ontologies with answering of CQs over sequences of ABoxes, resulting in a system that supplies rewritings for query answering over temporal fuzzy *DL-Lite*-ontologies.

1 Introduction

We report in this paper on work in progress regarding answering queries for extensions of the lightweight ontology language *DL-Lite* by fuzzy and also by temporal information, which are tailored towards the use in ontology-based situation recognition.

The main task in context-aware or self-adaptive systems is to recognize situations that might invoke an adaptation of the system to new conditions in their surroundings. To this end data is collected from multiple sources, often times sensors and stored in a data base system. A description of situations that might invoke an adaptation is matched against the data to detect the occurrence of a critical situation in the data. The ontology-based approach to such situation recognition enriches the observations made by sensors semantically and thus offers a higher-level view on the data collected. In the ontology-based approach the critical situations are captured by queries that are evaluated over the data enriched by the background knowledge captured in the ontology. Now, since potentially a huge amount of such preprocessed data has to be queried, efficient algorithms are expedient. In our case we employ the ontology language *DL-Lite* and answering of conjunctive queries (CQs), which are a simple form of first order queries, to recognize critical situations. It is well-known that *DL-Lite*-family of allows for answering of CQs in LogSpace [2, 7, 6], which is the same complexity

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

as answering database queries. This good complexity is achieved by the so-called *rewriting approach*, where the CQ is first enriched by the (relevant) information from the TBox and then this rewritten query is evaluated over a data base. The rewriting approach for answering CQs is implemented in optimized systems as QuOnto2 [1, 11], Ontop [13], Owlgres [14], and IQAROS [20] which perform well in practice.

Now, in context-aware systems that employ streams of sensor data, the expressivity of standard ontology languages often might be too limited. For instance, the numerical values obtained from the sensors are mapped to coarser logical categories such as high, medium or low. For such categories the concept membership of a particular measurement is rather vague calling for the use of fuzzy ontology languages that allow for *membership degrees* expressing that a particular measurement belongs to a category only to a certain extent. Another short-coming of standard ontology languages is the lack of modeling temporal information. Now, each of these two extensions are known to make reasoning in ontology languages undecidable, if allowed for modeling concepts. Thus, we chose combinations that allow for fuzzy or temporal information only in the data and in the query.

In our pragmatic approach to answering of (fuzzy) CQs, a crisp *DL-Lite* reasoner is used as a black box to obtain an initial rewriting of the CQ (without degrees). The obtained query gets extended in a second rewriting step by (1) fuzzy atoms, (2) degree variables that capture membership degrees, and (3) numerical predicates that realize the fuzzy operators. The resulting query can then be evaluated by a SQL engine. We have implemented this rewriting procedure in our reasoner FLite and report on its performance in this paper.

A similar extension of the classical rewriting approach has recently been employed for the temporal setting [5] and implemented [17]. In this setting the incoming information is modeled as a sequence of fact bases, one for each moment in time in which the system has been observed and thus realizing the sliding window approach. To describe situations, operators of the temporal logic LTL are admitted in the query. Such temporal operators can, for instance, express for a property that it is true at the next point in time or to be satisfied at some point in future. In this rewriting approach the temporal information queried for is treated by an additional rewriting step of the temporal query and the information is also retrieved by a query over the temporal database. Now, as it turns out both two-step rewriting methods can be combined in a straightforward, but elegant way. Yielding a method that is capable to answer CQs over temporal sequences that model data in a fuzzy way.

The rest of the paper is structured as follows: While in the next section we give the preliminaries on fuzzy *DL-Lite* and fuzzy ontologies, we describe the extension of the classical rewriting procedure to fuzzy information in Section 3 and its implementation in the FLite system. Section 4 gives results on a performance evaluation of FLite on a situation recognition use-case. Conclusions and future work end the paper in Section 6.

2 Preliminaries

We start with the concept language of *DL-Lite*$_R$ and then introduce our fuzzy variant of ABoxes. The information in a DL-knowledge base is described by means of the following atomic types: concept names form the set $\mathsf{N_C}$, role names from the set $\mathsf{N_R}$, individual names from the set $\mathsf{N_I}$ and for the queries also from the set of variables $\mathsf{N_V}$. From these elements the complex *DL-Lite*$_R$-concepts, -roles and -queries are constructed. *DL-Lite*$_R$-concepts and -roles are defined according to the following grammar:

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

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 form: $\langle B(a), \geqslant d\rangle$ 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 crisp in our setting. 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 here 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. Now, a *union of conjunctive queries* (UCQ) is simply a finite set of conjunctive queries that have the same number of answer variables.

The *semantics* of fuzzy *DL-Lite*$_R$ is provided an interpretation with an interpretation domain Δ and a mapping function that assigns values from the unit interval, instead of just 0 and 1 as in the classical case. More precisely, 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* $A^\mathcal{I} : \Delta^\mathcal{I} \rightarrow [0,1]$,
- $P \in \mathsf{N_R}$ to a *role membership function* $P^\mathcal{I} : \Delta^\mathcal{I} \times \Delta^\mathcal{I} \rightarrow [0,1]$.

The semantics of the complex concepts in fuzzy *DL-Lite*$_R$ is provided via the different families of fuzzy logic operators depicted in Table 1 and interpretations. 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:

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

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

Table 1. Families of fuzzy logic operators.

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

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 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:

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

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:

$$q_1(x, y) = \text{hasCPU}(x, y) \land \text{OverUsed}(y)$$
$$q_2(x) = \exists y \ \text{hasCPU}(x, y) \land \text{OverUsed}(y)$$

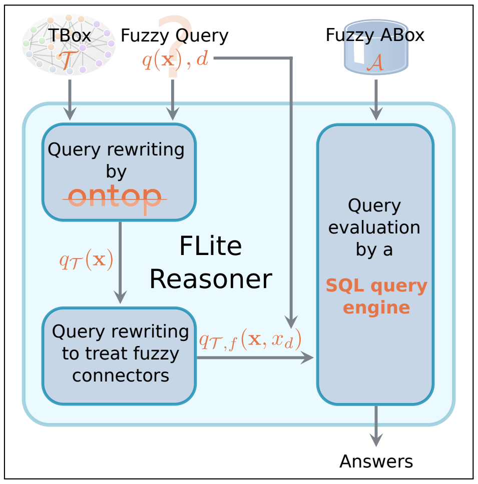
The query q_1 asks for pairs of Servers and CPUs with an overused CPU. The query q_2 asks for Servers, where the Server's CPU is overused. If conjunction and negation are interpreted as the Gödel family of operators, the certain answers w.r.t. $\mathcal{O}_{ex}$ are:

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

3 Fuzzy Query Answering by Extended Crisp Rewritings

In the following we are interested in answering the CQ $q(\mathbf{x})$, which is 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 [7, 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 concept 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 needed to realize the semantics of the fuzzy conjunction within a CQ. More precisely, it 'combines' the membership degrees obtained for the conjuncts to yield the membership degree of the conjunction. 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 $(d_1, \ldots, d_n) \in \Phi_\otimes$.



TBox
$\mathcal{T}$
Fuzzy Query
$q(\mathbf{x}), d$
Fuzzy ABox
$\mathcal{A}$
Query rewriting by ontop
$q_{\mathcal{T}}(\mathbf{x})$
FLite Reasoner
Query rewriting to treat fuzzy connectors
$q_{\mathcal{T},f}(\mathbf{x}, x_d)$
Query evaluation by a SQL query engine
Answers

Fig. 1. The FLite rewriting procedure.

For the CQ $q(\mathbf{x})$ to be answered, the two-step rewriting algorithm proceeds as follows:

1. The crisp *DL-Lite*$_R$ algorithm rewrites $q(\mathbf{x})$ to $q_{\mathcal{T}}(\mathbf{x})$ using the information from the TBox.

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 ABox $\mathcal{A}$ being 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 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:

$$\begin{aligned} q_{1\mathcal{T}_{ex}}(x, y) =& \{\text{hasCPU}(x, y) \land \text{OverUsed}(y)\} \\ q_{2\mathcal{T}_{ex}}(x) =& \{\exists y.\text{hasCPU}(x, y) \land \text{OverUsed}(y)\} \end{aligned}$$

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

$$\begin{aligned} q_{1\mathcal{T}_{ex}}^{f}(x, y, x_d) =& \{\text{hasCPU}(x, y, y_{d_1}) \land \text{OverUsed}(y, y_{d_2}) \land \Phi_\otimes(x_d, y_{d_1}, y_{d_2})\} \\ q_{2\mathcal{T}_{ex}}^{f}(x, x_d) =& \{\exists y.\text{hasCPU}(x, y, y_{d_1}) \land \text{OverUsed}(y, y_{d_2}) \land \Phi_\otimes(x_d, y_{d_1}, y_{d_2})\} \end{aligned}$$

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

$$\begin{aligned} ans(q_{1\mathcal{T}_{ex}}^{f}(x, x_d), \mathcal{A}_{ex}) =& \{(server_1, cpu_1, 0.6), (server_1, cpu_2, 0.8)\} \\ ans(q_{2\mathcal{T}_{ex}}^{f}(x, x_d), \mathcal{A}_{ex}) =& \{(server_1, 0.8)\}. \end{aligned}$$

The limitations of our pragmatic approach are explained in [9]. To sum up, this pragmatic approach yields sound and complete results for fuzzy semantics based on idempotent operators such as the Gödel family of operators. Non-idempotent operators may be simplified by highly optimized implementations such as Ontop. Consider $q_{\mathcal{T}}(x) := A(x) \land A(x)$ is simplified to $q_{\mathcal{T}}(x) := A(x)$, which is correct for crisp, but not for every fuzzy semantics. The correctness for the case of the Gödel family of operators 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. The latter does not apply for all other t-norms. The proposed methodology is complete but not sound for the Łukasiewicz and Gödel families of operators. Nevertheless, in [9], we devised a method by which each unsound answer can be identified and its correct degree estimated by an interval of membership values.

3.1 The FLite Reasoner Implementation

FLite[3] (Fuzzy *DL-Lite*$_R$ query engine) implements the above query answering algorithm and builds on the Ontop framework [13]. Implementation-wise, the rewriting procedure is a little more involved if a reasoner such as Ontop is deployed, since it operates on relational databases directly. 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 only virtual and implemented by 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 [12]. In this mapping, for the concept popularVideo each id of the table videos is 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, a video with an id of 12 and a popularity of 0.8 in the database 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 such fuzzy statements. It indicates that each element of the particular *column* (or SQL expression) is associated with a membership *degree*. Such a degree is either a column with values from $[0, 1]$, or an SQL expression corresponding to a fuzzy membership function. SQL expressions that contain the fuzzy marker function appear in the initial mapping $\mathcal{M}$ and in $q_{\mathcal{T}}(\mathbf{x})$ queries, while in $q_{\mathcal{T},f}(\mathbf{x}, x_d)$ queries these markers are converted to SQL expressions that return the membership degrees.

4 Performance Test of FLite

A Situation Recognition Application. 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

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

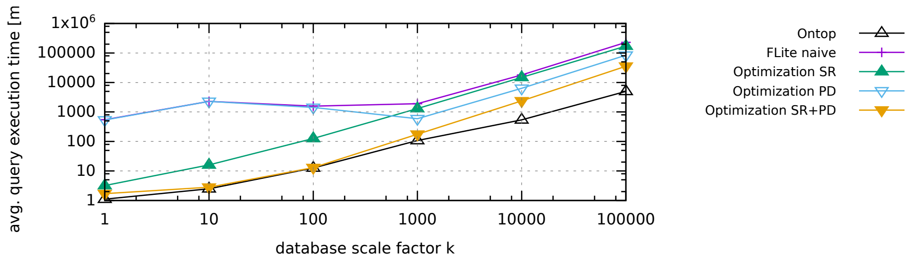


Fig. 2. Ontop is compared to optimized FLite versions SR, PD, and SR+PD. All setups were executed over a CQ with 13 atoms.

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.

We took this application to conduct a study of the performance of FLite. The background knowledge on hard- and software components of the HAEC system is modeled in a TBox which consists of 197 GCIs, 168 named classes and 38 roles (415 axioms in total). Each state of the HAEC system is stored in tables of a relational database, which store the information on the soft- and hardware of the HAEC system. The tables contain numerical values for boards, processes, requests, etc.[4] Our test 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.

FLite was evaluated over a series of databases of increasing size, i.e., scaled by a factor k. 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, a query with 13 atoms (2 crisp concepts, 7 fuzzy concepts, 4 crisp roles) in total was evaluated. 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 for increasing database size of Ontop with a crisp mapping compared to three variants of FLite (with Gödel t-norms and with fuzzy mapping): naive, optimization SR and optimization PD. Where SR means self join removal from the SQL statement and PD the pre-computation of membership degrees in the database.

Optimization SR shows run-time reduction up to a linear factor of about thousand, afterwards it converges with the naive FLite implementation. Thus,

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

optimizations for larger databases are necessary. Optimization PD in Figure 2 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.

Instead of comparing FLite with Ontop, a comparison with other fuzzy DL reasoners would have been desirable. However, reasoners such as LiFR [19], FuzzyDL [4], FiRE [15], and DeLorean [3] support only instance queries instead of conjunctive queries. Others such as the *DL-Lite* reasoner Ontosearch2 [10] or SoftFacts [16] could either not be obtained or installed.

5 Towards Temporal Fuzzy Query Answering on Streams

In temporal OBDA, one is not restricted to the knowledge about a single moment in time in a single ABox, but a *sequence of ABoxes* $\mathcal{A}_0, \ldots, \mathcal{A}_n$. A stream of data can be transformed into such a sequence by sampling the stream periodically and storing data together with a timestamp in a database. Such a complex entry in the database corresponds to an assertion in the ABox $\mathcal{A}_i$. This data can be used to recognize global contexts, through *temporal fuzzy conjunctive queries* (TFCQs), which refer to several point in time as well as to fuzzy information. A context-aware system is then able to subscribe and to transform data streams into a sequence of ABoxes and to evaluate a set of TFCQs periodically.

5.1 Introducing Temporal Query Answering

To accommodate the new information, we extend the notions of knowledge base, interpretation, and CQ to the temporal case. A *temporal knowledge base* (TKB) $\mathcal{K} = \langle \mathcal{O}, (\mathcal{A}_i)_{0 \leq i \leq n} \rangle$ consists of an ontology $\mathcal{O}$ and a finite sequence of ABoxes $\mathcal{A}_i$. Let $\mathfrak{I} = (\mathcal{I}_i)_{i \geq 0}$ be an infinite sequence of interpretations $\mathcal{I}_i = (\Delta, \cdot^{\mathcal{I}_i})$ over a non-empty domain Δ that is fixed (constant domain assumption). Then, $\mathfrak{I}$ is a *model* of $\mathcal{K}$ (written $\mathfrak{I} \models \mathcal{K}$) if

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

We next describe our query language, which allows to place LTL operators 'around' classical conjunctive queries. *Temporal conjunctive queries* (TCQs) are built from CQs as follows: (1) Every CQ is a TCQ and (2) If ϕ_1 and ϕ_2 are TCQs, then the following are also TCQs:

- $\phi_1 \wedge \phi_2$ (conjunction), $\phi_1 \vee \phi_2$ (disjunction),
- $\bigcirc \phi_1$ (next), $\bigcirc^- \phi_1$ (previous),
- $\phi_1 \, \mathsf{U}^< \, \phi_2$ (until), $\phi_1 \, \mathsf{S} \, \phi_2$(since),
- $\Box \phi_1$ (always), and $\Box^- \phi_1$ (always in the past).

ϕ	$(\mathfrak{a}(\phi))^{\mathfrak{I},i}$
A CQ ψ	$(\mathfrak{a}(\psi))^{\mathcal{I}_i}$
$\phi_1 \land \phi_2$	$(\mathfrak{a}_{\phi_1}(\phi_1))^{\mathfrak{I},i} \otimes (\mathfrak{a}_{\phi_2}(\phi_2))^{\mathfrak{I},i}$
$\phi_1 \lor \phi_2$	$(\mathfrak{a}_{\phi_1}(\phi_1))^{\mathfrak{I},i} \oplus (\mathfrak{a}_{\phi_2}(\phi_2))^{\mathfrak{I},i}$
$\bigcirc\phi_1$	If $i < n$, then $(\mathfrak{a}(\phi_1))^{\mathfrak{I},i+1}$, else 0.
$\bigcirc^-\phi_1$	If $i > 0$, then $(\mathfrak{a}(\phi_1))^{\mathfrak{I},i-1}$, else 0.
$\Box\phi_1$	$\otimes\{(\mathfrak{a}(\phi_1))^{\mathfrak{I},k} \mid i \leq k \leq n\}$
$\Box^-\phi_1$	$\otimes\{(\mathfrak{a}(\phi_1))^{\mathfrak{I},k} \mid 0 \leq k \leq i\}$
$\phi_1 \,\mathsf{U}\, \phi_2$	$\sup_{k,i \leq k \leq n} \otimes\{(\mathfrak{a}_{\phi_2}(\phi_2))^{\mathfrak{I},k}, (\mathfrak{a}_{\phi_1}(\phi_1))^{\mathfrak{I},j} \mid i \leq j < k\}$
$\phi_1 \,\mathsf{S}\, \phi_2$	$\sup_{k,0 \leq k \leq i} \otimes\{(\mathfrak{a}_{\phi_2}(\phi_2))^{\mathfrak{I},k}, (\mathfrak{a}_{\phi_1}(\phi_1))^{\mathfrak{I},j} \mid k < j \leq i\}$

Table 2. The semantics of TCQs.

As usual, we use the abbreviation $\Diamond\phi_1$ (eventually) for $\mathsf{true}\,\mathsf{U}\,\phi_1$; and, analogously for the past, $\Diamond^-\phi_1$, for $\mathsf{true}\,\mathsf{S}\,\phi_1$.
The *answers* to TCQs are mappings from the variables and individual names occurring in the query to elements of the domain of a given interpretation. We start by defining the semantics of CQs for Boolean queries as usual, through the notion of homomorphisms. Let $\psi(\boldsymbol{x}) = \exists \boldsymbol{y}.\psi'(\boldsymbol{x}, \boldsymbol{y})$ be a CQ, $\boldsymbol{a} = (a_1, \ldots, a_m)$ be a tuple of individual names, of same arity as the tuple $\boldsymbol{x}$, and $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation. A mapping π from the variables and individual names that occur in ψ to the elements of $\Delta^{\mathcal{I}}$ is a homomorphism of $\psi(a_1, \ldots, a_m)$ into $\mathcal{I}$ if

- $\pi(x_i) = \pi(a_i)$, for all $1 \leq i \leq m$;
- $\pi(a) = a^{\mathcal{I}}$, for all individual names a occurring in ψ;
- $\pi(t) \in A^{\mathcal{I}}$, for all concept atoms $A(t)$ in ψ; and
- $(\pi(t_1), \pi(t_2)) \in R^{\mathcal{I}}$, for all role atoms $R(t_1, t_2)$ in ψ.

Let now ϕ be a TCQ, $\mathfrak{a}$ be a mapping from the distinguished variables in ϕ to individual names, and $\mathfrak{I} = (\mathcal{I}_i)_{i \geq 0}$ be an infinite sequence of interpretations. We define the satisfaction relation $\mathfrak{I}, i \models \mathfrak{a}(\phi)$ by induction on the structure of ϕ based on Table 5.1; that is, $\mathfrak{I}, i\ models\langle\mathfrak{a}(\phi), d\rangle$ iff $(\mathfrak{a}(\phi))^{\mathfrak{I},i} \geq d$. Given a TKB $\mathcal{K}$, we say that $\mathfrak{a}$ is a *certain answer* to ϕ w.r.t. $\mathcal{K}$ at time point i, written $\mathcal{K}, i \models \mathfrak{a}(\phi)$, if we have $\mathfrak{I}, i \models \mathfrak{a}(\phi)$, for all models $\mathfrak{I}$ of $\mathcal{K}$.

5.2 An Algorithm for Answering TFCQs

Our algorithm to answer a TFCQ on fuzzy and temporal data is again a modified OBDA approach, which rewrites the given query into a standard database query. Here the rewritten query encodes the relevant ontological knowledge, the temporal conditions, and the membership variables, but addresses a general database. The algorithm to rewrite TFCQs is based on the following rewriting approach:

Assume, a TFCQ ϕ^t in TSPARQL is given, then the procedure $SQLify(\phi^t)$ rewrites ϕ^t into a SQL query (see Algorithm 1). First the algorithm stores the nested list of LTL-operators as they appear in the input TFCQ ϕ^t. Then the algorithm is applied to each sub-TFCQ in ϕ^t recursively until the considered sub-query is a plain CQ. This CQ ψ then undergoes the two-step rewriting procedure described in Section 3: first the CQ is *extended* by information from the ontology and then *fuzzified* by adding the membership degrees and degree variables to generate the corresponding membership degrees from the data. Subsequently, the extended and fuzzified CQs are recombined according to the nested list of temporal operators stored in the beginning.

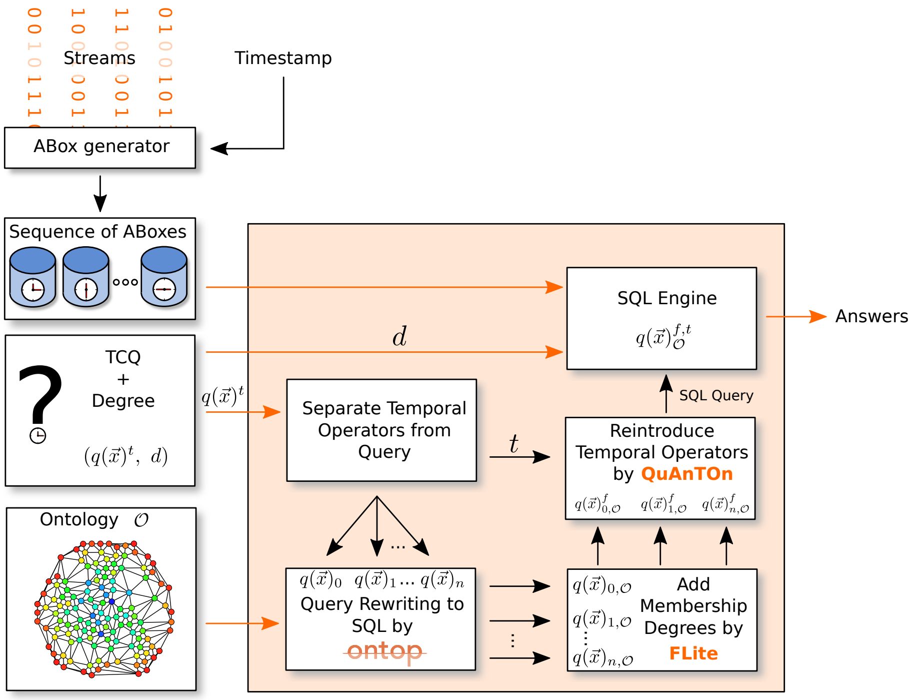
Algorithm 1 TFCQ to SQL rewriting

```
 1: function SQLIFY($\phi^t, \mathcal{T}$)
 2:     $\phi' \leftarrow \emptyset$
 3:     $t' \leftarrow temporal{-}operators(\phi^t)$   ▷ Store nested list of temporal LTL-operators
 4:     for all CQs $\phi$ in TCQ $\phi^t$ do
 5:         if $\phi$ contains temporal operator then
 6:             $\phi' \leftarrow append(\phi',\ SQLify(\phi, \mathcal{T}))$   ▷ Append a rewritten TFCQ to $\phi'$
 7:         else
 8:             $\phi_{\mathcal{T}} \leftarrow extend(\phi, \mathcal{T})$   ▷ Extends $\phi$ by information from $\mathcal{T}$
 9:             $\phi^f_{\mathcal{T}} \leftarrow fuzzify(\phi_{\mathcal{T}})$   ▷ Annotate $\phi_{\mathcal{T}}$ with fuzzy information
10:             $\phi' \leftarrow append(\phi',\ \phi^f_{\mathcal{T}}))$   ▷ Append the fuzzyfied CQ to $\phi'$
11:         end if
12:     end for
13:     $\phi^{t,f}_{\mathcal{T}} \leftarrow temporalize(\phi', t')$   ▷ Reintroduce LTL-operators from list into list of
    rewritten queries
14:     return $\phi^{t,f}_{\mathcal{T}}$
15: end function
```

Consider the query: $\phi^t = (\bigcirc^- \psi_1 \wedge (\bigcirc^- \bigcirc^- \psi_2))\, \mathsf{S}\, \psi_3$. Recall that the function $SQLify$ separates the temporal operators and thus splits the given query. As an example, we regard the rewriting $SQLify(\bigcirc^- \psi_1)$. Since ψ_1 is a plain CQ, it is extended and fuzzified as outlined above. The final rewriting obtained from the call $SQLify(\phi^t, \mathcal{T})$ can then be evaluated over a common relational database, and the obtained answers represent the answers to the query ϕ^t w.r.t. $\mathcal{T}$, with respect to fuzzy data.

5.3 Towards an Implementation

The idea for an implementation is to reuse existing tools to provide fast and efficient query answering even over large datasets. Techniques developed in FLite for fast query answering on fuzzy ontologies and in QuAnTOn [17] for temporal OBDA need to be combined in a single application, which allows for temporal OBDA over fuzzy data. Furthermore, an ABox generator needs to transform streaming data into the sequence of ABoxes, each with fuzzy information. An overview of the idea of such a system is given in Figure 3. The system input consists of (i) (possibly fuzzy) data referencing different time points, (ii) a pair (ϕ^t, d) containing a TFCQ ϕ^t and a degree d, and (iii) an ontology. The system then rewrites the query as described in the previous section and evaluates the rewritten query, $\phi_{\mathcal{T}}^{f,t}$, over a MySQL database. This evaluation yields a set of answer tuples with corresponding degrees. The system then returns those of these answers whose degree is $\geq d$.



Streams
Timestamp
ABox generator
Sequence of ABoxes
TCQ
+
Degree
$(q(\vec{x})^t,\ d)$
Ontology $\mathcal{O}$
$q(\vec{x})^t$
d
SQL Engine
$q(\vec{x})_{\mathcal{O}}^{f,t}$
Answers
SQL Query
Separate Temporal Operators from Query
t
Reintroduce Temporal Operators by QuAnTOn
$q(\vec{x})_{0,\mathcal{O}}^f$ $q(\vec{x})_{1,\mathcal{O}}^f$ $q(\vec{x})_{n,\mathcal{O}}^f$
$q(\vec{x})_0$ $q(\vec{x})_1$ ... $q(\vec{x})_n$
Query Rewriting to SQL by ~~ontop~~
$q(\vec{x})_{0,\mathcal{O}}$ $q(\vec{x})_{1,\mathcal{O}}$ $q(\vec{x})_{n,\mathcal{O}}$
Add Membership Degrees by FLite

Fig. 3. Schema of the implementation for temporal fuzzy query answering.

6 Conclusions

We have presented a pragmatic approach for answering (temporal) fuzzy conjunctive queries over (sequences of) fuzzy ABoxes and with respect to *DL-Lite*$_R$-ontologies. Our approach uses separate rewriting steps to incorporate ontological, fuzzy and temporal information. This allows to make use of standard query rewriting engines for the first step. Although described here for *DL-Lite*$_R$, our approach can be extended to other DLs that enjoy FOL rewritability. We have 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, albeit almost only linear increase of run-time for large ABoxes, when fuzzy information is queried. Furthermore, we presented an approach for temporal fuzzy query answering, which combines two rewriting-based query answering algorithms. A thorough investigation of this subject remains future 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. S. Borgwardt, M. Lippmann, and V. Thost. Temporalizing rewritable query languages over knowledge bases. *Journal of Web Semantics*, 2015. In press.
6. 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.
7. 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.
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, E. Thomas, and D. Sleeman. Ontosearch2: Searching and querying web ontologies. *Proc. of WWW/Internet*, 2006:211–218, 2006.
11. 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.
12. 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.
13. 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.
14. M. Stocker and M. Smith. Owlgres: A Scalable OWL Reasoner. In *OWLED*, volume 432, 2008.
15. G. Stoilos, N. Simou, G. Stamou, and S. Kollias. Uncertainty and the Semantic Web. *Intelligent Systems*, 21(5):84–87, 2006.
16. 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.
17. V. Thost, J. Holste, and O. Özçep. On implementing temporal query answering in *dl-lite* (extended abstract). In *Proceedings of the 28th International Workshop on Description Logics (DL-2015)*, Athens, Greece, 2015. CEUR Workshop Proceedings.
18. V. Thost, J. Holste, and Özgür Özçep. On implementing temporal query answering in *DL-Lite*. LTCS-Report 15-12, Chair for Automata Theory, TU Dresden, Germany, 2015. See http://lat.inf.tu-dresden.de/research/reports.html.
19. 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.

20. 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.