# Answering Vague Queries in Fuzzy DL-Lite

**Umberto Straccia**
ISTI- CNR, Italy
straccia@isti.cnr.it

## Abstract

*Fuzzy Description Logics* (fuzzy DLs) allow to describe structured knowledge with vague concepts. Unlike classical DLs, in fuzzy DLs an answer is a set of tuples ranked according to the degree they satisfy the query. In this paper, we consider fuzzy DL-Lite. We show how to compute efficiently the top-$k$ answers of a complex query (i.e. conjunctive queries) over a huge set of instances.

**Keywords:** Fuzzy Description Logics, top-$k$ query answering

## 1 Introduction

A substantial amount of work has been carried out in the context of *Description Logics* (DLs) [1]. DLs are a logical reconstruction of the so-called frame-based knowledge representation languages, with the aim of providing a simple well-established Tarski-style declarative semantics to capture the meaning of the most popular features of structured representation of knowledge. Nowadays, DLs have gained even more popularity due to their application in the context of the *Semantic Web* [9]. DLs play a particular role as they are essentially the theoretical counterpart of the *Web Ontology Language OWL DL*, the state of the art language to specify ontologies [9].

Fuzzy DLs [14, 16] extend classical DLs by allowing *fuzzy/imprecise concepts*. A major problem fuzzy DLs have to face with is that, unlike classical DLs, where an answer to a query is a set of tuples that satisfy a query, an answer is a set of tuples ranked according to the degree they satisfy the query. This poses a new challenge in case we have to deal with a huge amount of instances. Indeed, virtually every tuple of the knowledge base satisfies a query to a degree and, thus, has to be ranked. This is not feasible in practice.

We show how to compute efficiently the top-$k$ answers of a complex query (i.e. conjunctive queries) over a huge set of instances in fuzzy DL-Lite, a significant fuzzy DL, making the approach appealing for real world scenarios.

We first introduce the main notions related to fuzzy DLs and then define DL-Lite. In Section 3 we show how to answer queries. Section 4 concludes and outlines future research.

### 1.1 Basics of fuzzy DLs

DLs [1] are a family of logics for representing structured knowledge. Fuzzy DLs [14, 16] extend classical DLs by allowing to deal with fuzzy/imprecise concepts (for more on fuzzy DLs, see [14, 15, 16]). The main idea underlying fuzzy DLs is that an assertion $a{:}C$, stating that the constant $a$ is an instance of concept $C$, rather being interpreted as either true or false, will be mapped into a truth value $n \in [0, 1]$. The intended meaning is that $n$ indicates to which extend '$a$ is a $C$'. For illustrative purposes, we recall here fuzzy $\mathcal{ALC}$ (see Table 1). *Concepts* and *roles* denote unary and binary predicates respectively. From a syntax point of view, concept forming operators allow to build more complex concepts, starting from so-called *atomic* concepts. *Fuzzy assertions* allow to state that an

| Concepts | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Syntax* | | | | *Semantics* | | | *Examples* |
| $C, D$ | $\longrightarrow$ | $\top$ | (top concept) | $\top^{\mathcal{I}}(x)$ | $=$ | $1$ | |
| | | $\bot$ | (bottom concept) | $\bot^{\mathcal{I}}(x)$ | $=$ | $0$ | |
| | | $A$ | (atomic concept) | $A^{\mathcal{I}}(x)$ | $\in$ | $[0,1]$ | `Human` |
| | | $C \sqcap D$ | (concept conjunction) | $(C_1 \sqcap C_2)^{\mathcal{I}}(x)$ | $=$ | ${C_1}^{\mathcal{I}}(x) \wedge {C_2}^{\mathcal{I}}(x))$ | `Human` $\sqcap$ `Male` |
| | | $C \sqcup D$ | (concept disjunction) | $(C_1 \sqcup C_2)^{\mathcal{I}}(x)$ | $=$ | ${C_1}^{\mathcal{I}}(x) \vee {C_2}^{\mathcal{I}}(x))$ | `Nice` $\sqcup$ `Rich` |
| | | $\neg C$ | (concept negation) | $(\neg C)^{\mathcal{I}}(x)$ | $=$ | $\neg C^{\mathcal{I}}(x))$ | $\neg$`Male` |
| | | $\exists R.C$ | (existential quantification) | $(\exists R.C)^{\mathcal{I}}(x)$ | $=$ | $\sup_{y \in \Delta^{\mathcal{I}}}\{R^{\mathcal{I}}(x,y) \wedge C^{\mathcal{I}}(y))\}$ | $\exists$`has_child.Blond` |
| | | $\forall R.C$ | (universal quantification) | $(\forall R.C)^{\mathcal{I}}(x)$ | $=$ | $\inf_{y \in \Delta^{\mathcal{I}}}\{R^{\mathcal{I}}(x,y) \Rightarrow C^{\mathcal{I}}(y)\}$ | $\forall$`has_child.Human` |

**Fuzzy assertion:** is of the form $\langle a{:}C, n\rangle$, with semantics $\mathcal{I} \models \langle a{:}C, n\rangle$ iff $C^{\mathcal{I}}(a^{\mathcal{I}}) \geq n$. An example is $\langle$`John`:`Happy_Father`$, 0.7\rangle$. Related to roles, we have assertions $\langle (a,b){:}R, n\rangle$ with semantics $\mathcal{I} \models \langle (a,b){:}R, n\rangle$ iff $R^{\mathcal{I}}(a^{\mathcal{I}}, b^{\mathcal{I}}) \geq n$. An example is $\langle$(`John`, `Mary`):`loves`$, 0.6\rangle$.

**Inclusion axiom:** is of the form $C \sqsubseteq D$, with semantics $\mathcal{I} \models C \sqsubseteq D$ iff $\forall x \in \Delta^{\mathcal{I}}.C^{\mathcal{I}}(x) \leq D^{\mathcal{I}}(x)$. An example is, `Happy_Father` $\sqsubseteq$ `Man` $\sqcap \exists$`has_child.Female`

**Knowledge base:** is of the form $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$, where $\mathcal{T}$ ($\mathcal{A}$) is a finite set of inclusion axioms (fuzzy assertions). The semantics is, $\mathcal{I} \models \mathcal{K}$ iff $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \models \mathcal{A}$.

Table 1: fuzzy $\mathcal{ALC}$.

individual is an instance of concept at least to a given degree in $[0,1]_{\mathbb{Q}} = [0,1] \cap \mathbb{Q}$ (a rational number in $[0,1]$). Similarly, for roles. *Inclusion axioms* allow to state inclusion relationships among concepts.

From a semantics point o view, *interpretations* map concept (resp. roles) into functions over the domain $\Delta^{\mathcal{I}}$ (resp. $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$) into $[0,1]_{\mathbb{Q}}$. The interpretation of complex concepts is given in terms of so-called t-norm (interpreting conjunction), s-norms (interpreting disjunction), negation function (interpreting negation), and implication function (interpreting implication) [8]. For the sake of this paper, we fix the semantics to the so-called "Zadeh semantics" (see Table 1), which is widely used for fuzzy DLs. That is, $x \wedge y = \min(x,y)$, $x \vee y = \max(x,y)$, $\neg x = 1 - x$ and $x \Rightarrow y = \neg x \vee y$.

Given a fuzzy KB $\mathcal{K}$, and a fuzzy assertion $\alpha$ (resp. an inclusion axiom $C \sqsubseteq D$), we say that $\mathcal{K}$ *entails* $\alpha$ (resp. $C \sqsubseteq D$), denoted $\mathcal{K} \models \alpha$ (resp. $\mathcal{K} \models C \sqsubseteq D$), iff each model of $\mathcal{K}$ satisfies $\alpha$ (resp. $C \sqsubseteq D$). Finally, given $\mathcal{K}$ and a fuzzy assertion $\alpha$, it is of interest to compute $\alpha$'s best lower and upper truth value bounds. The *greatest lower bound* of $\alpha$ w.r.t. $\mathcal{K}$ (denoted $glb(\mathcal{K}, \alpha)$) is $glb(\mathcal{K}, \alpha) = \sup\{n \mid \mathcal{K} \models \langle \alpha, n\rangle\}$ where $\sup \emptyset = 0$. Determining the *glb* is called the *Best Truth Value Bound* (BTVB) problem. Finally, the basic inference problems are:

**Consistency:** Check if a fuzzy KB is consistent, i.e. has a model?

**Subsumption:** structure knowledge, compute the taxonomy, i.e. $\mathcal{K} \models C \sqsubseteq D$?

**Entailment:** Check if $a$ is instance of $C$ to degree $\geq n$, i.e. $\mathcal{K} \models \langle a{:}C, n\rangle$?

**BTVB:** Best Truth Value Bound problem, i.e. determine $glb(\mathcal{K}, a{:}C) = \sup\{n \mid \mathcal{K} \models \langle a{:}C, n\rangle\}$?

**Fuzzy retrieval:** Retrieve the top-$k$ ranked constants that instantiate $C$ w.r.t. the best truth value bound, i.e. find the top-$k$ ranked constants of the set $ans(\mathcal{K}, C) = \{\langle a, glb(\mathcal{K}, a{:}C)\rangle\}$.

In [14] decision procedures for the satisfiability, the entailment and the BTVB problem are given for fuzzy $\mathcal{ALC}$, but with the restrictions on the form of terminological axioms and terminologies. Also, "Zadeh semantics" is used for interpreting the connectives. In [15] a more efficient method is presented for the BTVB problem and covers Lukasiewicz semantics as well. Also the language is more expressive than fuzzy $\mathcal{ALC}$ as it allows to explicitly represent concept membership functions and concept modifiers.

However, a major drawback of current existing reasoning algorithm for fuzzy $\mathcal{ALC}$ is their inefficiency to solve the fuzzy retrieval problem. To date, we have to compute for all constants in the knowledge base its best truth value bound, then order the constants according to this degree bound and then select the top-$k$ ranked constants. This is clearly not a feasible solution if the KB deals with a huge amount of constants.

This will be addressed in the next section, but we have to rely on a simpler language than

fuzzy $\mathcal{ALC}$. But, on the other hand we allow for more complex queries.

## 2 Fuzzy DL-Lite

To come up with an efficient solution to the fuzzy retrieval problem, we propose to consider a sub-language of fuzzy $\mathcal{ALC}$. Indeed, we consider fuzzy DL-Lite, the fuzzy variant of DL-Lite [4], which has been proposed as a computationally tractable (in data complexity) DL to query large crisp databases. In fuzzy DL-Lite, concepts are defined as follows:

$$\begin{array}{lcl} B & \rightarrow & A \mid \exists R \\ C & \rightarrow & B \mid \neg B \mid C_1 \sqcap C_2 \\ R & \rightarrow & P \mid P^- \end{array}$$

where $A$ denotes an atomic concept, $P$ denotes an atomic role. A role $R$ can be either an atomic role $P$ or its *inverse* $P^-$. $B$ denotes a *basic concept* that can be either an atomic concept, a concept of the form $\exists R$, i.e. the standard DL construct of unqualified existential quantification (equivalent to $\exists R.\top$). $C$ denotes a *general concept*. Note that we use negation on basic concepts only, and we do not allow for disjunction.

A fuzzy DL-Lite *knowledge base* is pair $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ and $\mathcal{A}$ are finite sets of fuzzy DL-Lite axioms and assertions. A *fuzzy DL-Lite axiom* is of the form

$$\begin{array}{ll} B \sqsubseteq C & \text{(inclusion axiom)} \\ \texttt{fun}(R) & \text{(functionality axiom)} \end{array}$$

(functionality axiom expresses the functionality of a role), while a *fuzzy assertion* is of the form $\langle a{:}B, n \rangle$ (*fuzzy concept assertion*), or of the form $\langle (a,b){:}P, n \rangle$ (*fuzzy role assertion*). Additionally, without loss of generality, we assume that if $\langle a{:}B, n \rangle$ and $\langle a{:}B, m \rangle$ belong to a fuzzy KB, then $n = m$ (otherwise, we just discard the fuzzy concept assertion with the lower truth degree). Similarly for fuzzy role assertions.

Our language allows for querying the extensional knowledge of a KB in a much more powerful way than usual fuzzy DLs. Specifically, fuzzy DL-Lite allows for using conjunctive queries of arbitrary complexity. A *conjunctive query* $q$ over a knowledge base $\mathcal{K}$ is an expression of the form

$$q(\mathbf{x}) \leftarrow \exists \mathbf{y}.conj(\mathbf{x}, \mathbf{y})$$

where $\mathbf{x}$ are the *distinguished variables*, $\mathbf{y}$ are existentially quantified variables called the *non-distinguished variables*, and $conj(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms of the form $B(z)$, or $P(z_1, z_2)$, where $B$ and $P$ are respectively a basic concept and a role (but, not inverse role) in $\mathcal{K}$, and $z, z_1, z_2$ are constants in $\mathcal{K}$ or variables in $\mathbf{x}$ or $\mathbf{y}$. Note that the use of conjunctive queries in classical DLs is not novel as it has already been used e.g. in DL-Lite. We extend this idea to the fuzzy context.

The semantics of fuzzy DL-Lite is similar to fuzzy $\mathcal{ALC}$, and is given in terms of interpretations. The major difference is that we consider a *fixed infinite domain* $\Delta$ [1]. We assume to have one object for each constant, denoting exactly that object. In other words, we have standard names [10], and we will not distinguish between the alphabet of constants and $\Delta$. So, an *interpretation* is now $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ and consists of a fixed infinite domain $\Delta$ with an interpretation function $\cdot^{\mathcal{I}}$ mapping concepts and roles as for fuzzy $\mathcal{ALC}$, where additionally $(P^-)^{\mathcal{I}}(x,y) = P^{\mathcal{I}}(y,x)$ and for functional roles we say that a fuzzy interpretation satisfies the axiom $\texttt{fun}(R)$ iff for all $x \in \Delta$ there is an unique $y \in \Delta$ such that $R^{\mathcal{I}}(x,y) > 0$. Finally, we extend fuzzy interpretations to queries $q(\mathbf{x}) \leftarrow \exists \mathbf{y}.conj(\mathbf{x}, \mathbf{y})$ as follows. Of course, for a basic concept $B$ (role $R$) appearing in the body of the rule, the interpretation of $B$ ($R$) is $B^{\mathcal{I}}$ ($R^{\mathcal{I}}$). For $\mathbf{c}, \mathbf{c}' \in \Delta \times \ldots \times \Delta$, the conjunction $conj(\mathbf{c}, \mathbf{c}')$ is interpreted as the t-norm (in our case, min) of the conjuncts, while the existential quantifier is interpreted as sup. Therefore, for $\mathbf{c} \in \Delta \times \ldots \times \Delta$ the degree of truth of $\exists \mathbf{y}.conj(\mathbf{x}, \mathbf{y})$ under $\mathcal{I}$ is $\sup_{\mathbf{c}' \in \Delta} \{(conj(\mathbf{c}, \mathbf{c}'))^{\mathcal{I}}\}$. This is also the truth of $q^{\mathcal{I}}(\mathbf{c})$, i.e., for all $\mathbf{c} \in \Delta \times \ldots \times \Delta$,

$$q^{\mathcal{I}}(\mathbf{c}) = \sup_{\mathbf{c}' \in \Delta} \{(conj(\mathbf{c}, \mathbf{c}'))^{\mathcal{I}}\} .$$

Then we say that: $(i)$ a fuzzy DL-Lite knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ *entails* $q(\mathbf{c})$ to degree $n$, written $\mathcal{K} \models \langle q(\mathbf{c}), n \rangle$ iff for any model $\mathcal{I}$ of $\mathcal{K}$, $q^{\mathcal{I}}(\mathbf{c}) \geq n$; and of course, $(ii)$ the *greatest lower bound* of $q(\mathbf{c})$ w.r.t. $\mathcal{K}$ (denoted $glb(\mathcal{K}, q(\mathbf{c}))$) is

$$glb(\mathcal{K}, q(\mathbf{c})) = \sup\{n \mid \mathcal{K} \models \langle q(\mathbf{c}), n \rangle\} .$$

[1] The domain has to be infinite as pointed out in [4]. In particular, it can be shown that, in general, given a DL-Lite knowledge base $\mathcal{K}$, there exists no finite structure $S$ such that, for every conjunctive query $q$, the set of answers to $q$ over $\mathcal{K}$ is the result of evaluating $q$ itself over $S$ (see [3]).

As fuzzy DL-Lite deals with conjunctive queries, the basic reasoning services that mainly concerns us is the fuzzy knowledge base satisfiability problem and the fuzzy retrieval problem, where this latter is defined as:

**Fuzzy retrieval:** Retrieve the top-$k$ ranked tuples $\mathbf{c}$ that instantiate the query $q$ w.r.t. the best truth value bound, i.e. find the top-$k$ ranked tuples of the answer set of $q$, denoted

$$ans_k(\mathcal{K}, q) = \mathtt{Top}_k(\{\langle \mathbf{c}, glb(\mathcal{K}, q(\mathbf{c}))\rangle\}) \ .$$

**Example 1** *Suppose we have information about hotels and conference locations and the distance between them, as shown in the table below. Assume we have a function, which measures the closeness degree between hotels and conference locations, depending on the distance. We may ask to find hotels close to a conference location, i.e. rank the hotels according to their degree of closeness.*

$$\begin{array}{rcl} \mathtt{Hotel} & \sqsubseteq & \exists \mathtt{hasLoc} \\ \mathtt{Conference} & \sqsubseteq & \exists \mathtt{hasLoc} \\ \mathtt{Hotel} & \sqsubseteq & \neg \mathtt{Conference} \\ Close(hl, cl) & = & \max(0, 1 - \frac{\mathtt{distance(hl.hasLoc, cl.hasLoc)}}{1000}) \end{array}$$

| HotelID | hasLoc |
|---|---|
| h1 | hl1 |
| h2 | hl2 |
| ⋮ | ⋮ |

| ConferenceID | hasLoc |
|---|---|
| c1 | cl1 |
| c2 | cl2 |
| ⋮ | ⋮ |

| hasLoc | hasLoc | distance |
|---|---|---|
| hl1 | cl1 | 300 |
| hl1 | cl2 | 500 |
| hl2 | cl1 | 750 |
| hl2 | cl2 | 800 |
| ⋮ | ⋮ | |

| hasLoc | hasLoc | close |
|---|---|---|
| hl1 | cl1 | 0.7 |
| hl1 | cl2 | 0.5 |
| hl2 | cl1 | 0.25 |
| hl2 | cl2 | 0.2 |
| ⋮ | ⋮ | |

*We may express our information need using the query (*c1 *is our conference location)*

$$q(h) \leftarrow \mathtt{hasLocation}(h, hl) \wedge \mathtt{hasLocation}(\mathtt{c1}, cl) \wedge \mathtt{close}(hl, cl) \ .$$

*Then we want to retrieve the top-k answers, according to the degree of closeness. It is not feasible to compute all degrees first and then rank them (there may be a huge amount of hotels and conference locations).*

Despite the simplicity of its language and the specific form of inclusion axioms allowed, fuzzy DL-Lite is able to capture the main notions (though not all, obviously) of both ontologies, and of conceptual modelling formalisms used in databases and software engineering (i.e., ER and UML class diagrams). In particular, fuzzy DL-Lite axioms allow us to specify ISA, e.g., stating that concept $A_1$ is subsumed by concept $A_2$, using $A_1 \sqsubseteq A_2$; *disjointness*, e.g., between concepts $A_1$ and $A_2$, using $A_1 \sqsubseteq \neg A_2$; *role-typing*, e.g., stating that the first (resp., second) component of the relation $R$ is an instance of $A_1$ (resp., $A_2$), using $\exists P \sqsubseteq A_1$ (resp., $\exists P^- \sqsubseteq A_2$); *participation constraints*, e.g., stating that all instances of concept $A$ participate to the relation $P$ as the first (resp., second) component, using $A \sqsubseteq \exists P$ (resp., $A \sqsubseteq \exists P^-$); *non-participation constraints*, using $A \sqsubseteq \neg\exists P$ and $A \sqsubseteq \neg\exists P^-$; *functionality restrictions* on relations, using $\mathtt{fun}(R)$. Additionally, observe that fuzzy DL-Lite does allow for cyclic axioms. A calculus for inclusion axioms is yet unknown for more expressive fuzzy DLs such as fuzzy $\mathcal{ALC}$. Notice that fuzzy DL-Lite is a strict subset of fuzzy OWL Lite and, thus of OWL DL [9], which presents some constructs (e.g., some kinds of role restrictions) that are non expressible in fuzzy DL-Lite(and that make reasoning in fuzzy OWL Lite non-tractable in general).

## 3 Query Answering

We discuss reasoning in fuzzy DL-Lite. We concentrate on the basic reasoning task in the context of using ontologies to access large data repositories. Namely that of answering (conjunctive) queries over a fuzzy DL-Lite knowledge base $\mathcal{K}$. To this end, we first have to check if $\mathcal{K}$ is satisfiable, as querying an inconsistent KB does not make sense in our case. We provide a simple method for checking satisfiability of $\mathcal{K}$. Second, we present an efficient top-$k$ query answering procedure.

The main feature of fuzzy DL-Lite is that we can almost follow the reasoning part of it's crisp counterpart DL-Lite. So, we start by preparing our knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$ for effective management. That means, we first normalize it into a suitable form and then store the data in $\mathcal{A}$ into a relational database.

The *normalization* of $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is obtained by transforming $\mathcal{K}$ as follows. $\mathcal{A}$ is expanded by adding to $\mathcal{A}$ the assertions $\langle a{:}\exists P, n\rangle$ and $\langle b{:}\exists P^-, n\rangle$ for each $\langle (a, b){:}P, n\rangle \in \mathcal{A}$. Con-

cerning $\mathcal{T}$, we split concept conjunctions using the rule: if $\mathcal{T}$ contains $B \sqsubseteq C_1 \sqcap C_2$, then replace it with the two axioms $B \sqsubseteq C_1$ and $B \sqsubseteq C_2$. This is possible as for any interpretation $\mathcal{I}$, $\mathcal{I} \models B \sqsubseteq C_1 \sqcap C_2$ iff $\mathcal{I} \models B \sqsubseteq C_1$ and $\mathcal{I} \models B \sqsubseteq C_2$ (note that this is not true in e.g. Lukasiewicz logic). Now $\mathcal{T}$ contains axioms of the form $(i)$ $B_1 \sqsubseteq B_2$, where $B_1$ and $B_2$ are basic concepts (i.e., each of them is either an atomic or an existential concept), which we call *positive inclusions* (PIs); $(ii)$ $B_1 \sqsubseteq \neg B2$, where $B_1$ and $B_2$ are basic concepts, which we call *negative inclusions* (NIs); $(iii)$ functionality axioms on roles of the form $\texttt{fun}(P)$ or $\texttt{fun}(P^-)$.

Then $\mathcal{T}$ is expanded by computing all (non-trivial) NIs between basic concepts implied by $\mathcal{T}$. More precisely, $\mathcal{T}$ is closed with respect to the following inference rule: if $B_1 \sqsubseteq B_2$ occurs in $\mathcal{T}$ and either $B_2 \sqsubseteq \neg B_3$ or $B_3 \sqsubseteq \neg B_2$ occurs in $\mathcal{T}$ (where $B_1, B_2, B_3$ are arbitrary basic concepts), then add $B_1 \sqsubseteq \neg B_3$ to $\mathcal{T}$. It can be shown that, after the above closure of $\mathcal{T}$ , for every pair of basic concepts $B_1, B_2$, we have that $\mathcal{T} \models B_1 \sqsubseteq \neg B_2$ iff either $B_1 \sqsubseteq \neg B_2 \in \mathcal{T}$ or $B_2 \sqsubseteq \neg B_1 \in \mathcal{T}$. This is not surprising as the law of contraposition holds: $x \Rightarrow y \equiv \neg x \Rightarrow \neg y$ [8]. Finally, it is easy to show that the normalization process transforms $\mathcal{K}$ into a model preserving form. In the following, without loss of generality we assume that every concept name or role name occurring in $\mathcal{A}$ also occurs in $\mathcal{T}$.

Once $\mathcal{K}$ is normalized, we store it under the control of a Data Base Management System (DBMS), in order to effectively manage and retrieve constants. To this aim, we construct a relational database which faithfully represents a normalized ABox $\mathcal{A}$. More precisely, $(i)$ for each basic concept $B$ occurring in $\mathcal{A}$, we define a relational table $tab_B$ of arity 2, such that $\langle a, n\rangle \in tab_B$ iff $\langle a{:}B, n\rangle \in \mathcal{A}$; and $(ii)$ for each role $P$ occurring in $\mathcal{A}$, we define a relational table $tab_P$ of arity 3, such that $\langle a, b, n\rangle \in tab_P$ iff $\langle (a, b){:}P, n\rangle \in \mathcal{A}$. We denote with $\texttt{DB}(\mathcal{A})$ the relational database thus constructed. We assume that the tuples are stored in $\texttt{DB}(\mathcal{A})$ in decreasing order according to the truth degree weight.

**KB satisfiability.** To check the satisfiability of a normalized fuzzy KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, we verify the following conditions: $(i)$ there exists a NI $B_1 \sqsubseteq \neg B_2 \in \mathcal{T}$ and a constant $a$ such that the fuzzy assertions $\langle a{:}B_1, n\rangle$ and $\langle a{:}B_2, m\rangle$ belong to $\mathcal{A}$ with $n + m > 1$; $(ii)$ there exists an axiom $\texttt{fun}(P)$ (respectively, $\texttt{fun}(P^-)$) in $\mathcal{T}$ and three constants $a, b, c$ such that both $\langle (a, b){:}P, n\rangle$ and $\langle (a, c){:}P, m\rangle$ (resp., $\langle (b, a){:}P, n\rangle$ and $\langle (c, a){:}P, m\rangle$) belong to $\mathcal{A}$ with $n, m > 0$. Informally, the first condition corresponds to checking whether $\mathcal{A}$ explicitly contradicts some NI in $\mathcal{T}$, and the second condition corresponds to check whether $\mathcal{A}$ violates some functionality axiom in $\mathcal{T}$. If one of the conditions below holds, then $\mathcal{K}$ is not satisfiable. Otherwise, $\mathcal{K}$ is satisfiable.

Interestingly, the algorithm verifies such conditions by posing to $\texttt{DB}(\mathcal{A})$ simple SQL queries. Notice that the algorithm does not consider the PIs occurring in $\mathcal{T}$ during its execution. Indeed, it can be shown as in [4] that PIs do not affect the consistency of a fuzzy DL-Lite KB, if $\mathcal{T}$ is normalized. Also, note that we do not require to take PI axioms into account. For instance, if $\mathcal{T} = \{B_1 \sqsubseteq B_2\}$, $\mathcal{A} = \{\langle a{:}B_1, n\rangle, \langle a{:}B_2, m\rangle\}$ with $n > m$ is satisfiable, as $\langle \mathcal{T}, \mathcal{A}\rangle \models \langle a{:}B_2, n\rangle$.

**Query answering.** We now describe our query answering procedure. It closely follows [4]. The process is divided into two steps:

1. By considering $\mathcal{T}$ only, the user query $q$ is *reformulated* into a set of conjunctive queries $r(q, \mathcal{T})$. Informally, the basic idea is that the reformulation procedure closely resembles a top-down resolution procedure for logic programming, where each inclusion axiom $B_1 \sqsubseteq B_2$ is seen as a logic programming rule of the form $B_2(x) \leftarrow B_1(x)$. For instance, given the query $q(x) \leftarrow B(x) \wedge A(x)$ and suppose that $\mathcal{T}$ contains the inclusion axioms $B_1 \sqsubseteq A$ and $B_2 \sqsubseteq A$, then we can reformulate the query into two queries $q(x) \leftarrow B(x) \wedge B_1(x)$ and $q(x) \leftarrow B(x) \wedge B_2(x)$, exactly as it happens for top-down resolution methods for logic programming.
2. The reformulated queries in $r(q, \mathcal{T})$ are evaluated over $\mathcal{A}$ only (considered as a database), producing the requested answer $ans_k(\mathcal{K}, q)$. For instance, for the previous query, the answers will be the union of the answers pro-

duced by all three queries.

In the following, we illustrate our approach from a technical point of view.

We start with the query reformulation step. We say that an argument of an atom in a query is *bound* if it corresponds to either a distinguished variable or a shared variable, i.e., a variable occurring at least twice in the query body, or a constant, while we say that it is *unbound* if it corresponds to a non-distinguished non-shared variable (as usual, we use the symbol "_" to represent non-distinguished non-shared variables). Notice that, an atom of the form $(\exists P)(x)$ (resp. $(\exists P^-)(x)$) has the same meaning as $P(x, \_)$ (resp. $P(\_, x)$). For ease of exposition, in the following we will use the latter form only. A PI axiom $\tau$ is *applicable* to an atom $B(x)$, if $\tau$ has $B$ in its right-hand side, and $I$ is applicable to an atom $P(x_1, x_2)$, if either $(i)$ $x_2 = \_$ and the right-hand side of $\tau$ is $\exists P$, or $(ii)$ $x_1 = \_$ and the right-hand side of $\tau$ is $\exists P^-$. Roughly speaking, an inclusion $\tau$ is applicable to an atom $g$ if all bound arguments of $g$ are propagated by $\tau$. Obviously, since all PIs in $\mathcal{T}$ are unary, they are never applicable to atoms with two bound arguments. We indicate with $gr(g; \tau)$ the atom obtained from the atom $g$ by applying the inclusion axiom $\tau$, i.e., if $g = B_1(x)$ (resp., $g = P_1(x, \_)$ or $g = P_1(\_, x)$) and $\tau = B_2 \sqsubseteq B_1$ (resp., $\tau = B_2 \sqsubseteq \exists P_1$) or $\tau = B_2 \sqsubseteq \exists P_1^-$), we have:

- $gr(g, \tau) = P_2(x, \_)$, if $B_2 = \exists P_2$;
- $gr(g, \tau) = P_2(\_, x)$, if $B_2 = \exists P_2^-$;
- $gr(g, \tau) = A(x)$, if $B_2 = A$, where $A$ is a basic concept.

We are now ready to recall the query reformulation algorithm `QueryRef` [4]. Given a conjunctive query $q$ and a set of axioms $\mathcal{T}$, the algorithm reformulates $q$ in terms of a set of queries $r(q, \mathcal{T})$, which then can be evaluated over $\mathtt{DB}(\mathcal{A})$.

In the algorithm, $q[g/g']$ denotes the query obtained from $q$ by replacing the atom $g$ with a new atom $g'$. Informally, the algorithm first reformulates the atoms of each query $q \in S$,

**Algorithm 1** `QueryRef`$(q, \mathcal{T})$

**Input:** Conjunctive query $q$, fuzzy DL-Lite axioms $\mathcal{T}$.
**Output:** Set of reformulated conjunctive queries $r(q, \mathcal{T})$.

```
1: r(q, T) := {q}
2: repeat
3:     S = r(q, T)
4:     for all q ∈ S do
5:         for all g ∈ q do
6:             if τ ∈ T is applicable to g then
7:                 r(q, T) := r(q, T) ∪ {q[g/gr(g, τ)]}
8:         for all g1, g2 ∈ q do
9:             if g1 and g2 unify then
10:                r(q, T) := r(q, T) ∪ {κ(reduce(q, g1, g2))}
11: until S = r(q, T)
12: return r(q, T)
```

and produces a new query for each atom reformulation (step 5.). Roughly speaking, PIs are used as rewriting rules, applied from right to left, that allow to compile away in the reformulation the knowledge of $\mathcal{T}$ that is relevant for answering $q$.

At step 8., for each pair of atoms $g_1, g_2$ that unify, the algorithm computes the query $q' = reduce(q, g_1, g_2)$, by applying to $q$ the most general unifier between $g_1$ and $g_2$ [2]. Due to the unification, variables that were bound in $q$ may become unbound in $q'$. Hence, PIs that were not applicable to atoms of $q$, may become applicable to atoms of $q'$ (in the next executions of step (5)). Function $\kappa$ applied to $q'$ replaces with _ each unbound variable in $q'$. This concludes the query reformulation step. In the following, we show with a small example the behaviour of the query reformulation algorithm.

**Example 2** *Suppose the set of inclusion axioms is* $\mathcal{T} = \{\exists P_2^- \sqsubseteq A, A \sqsubseteq \exists P_1, B \sqsubseteq \exists P_2\}$. *We also assume that the set of assertions* $\mathcal{A}$ *is stored in the tables below ($P_2$ is a role, while $B$ is a basic concept):*

| $P_2$ | | |
|---|---|---|
| a | s | 1.0 |
| b | t | 0.8 |
| c | q | 0.6 |
| d | q | 0.4 |

| $B$ | |
|---|---|
| e | 0.9 |
| f | 0.7 |
| g | 0.5 |
| h | 0.3 |

*Assume our query is* $q_0$*:*

$$q(x) \leftarrow P_2(x, y) \wedge P_1(y, \_) \ .$$

*Then at the first execution of step 7., the algorithm inserts query* $q_1$, $q(x) \leftarrow P_2(x, y) \wedge$

[2] We say that two atoms $g_1 = r(x_1, \ldots, x_n)$ and $g_2 = r(y_1, \ldots, y_n)$ *unify*, if for all $i$, either $x_i = y_i$ or $x_i = \_$ or $y_i = \_$. If $g_1$ and $g_2$ unify, then the unification of $g_1$ and $g_2$ is the atom $r(z_1, \ldots, z_n)$, where $z_i = x_i$ if $x_i = y_i$ or $y_i = \_$, otherwise $z_i = y_i$ (see, [3]).

*$A(y)$ into $r(q, \mathcal{T})$ using the axiom $A \sqsubseteq \exists P_1$. At the second execution of step 7., the algorithm inserts query $q_2$, $q(x) \leftarrow P_2(x, y) \wedge P_2(\_, y)$ using the axiom $\exists P_2^- \sqsubseteq A$. Since the two atoms of the second query unify, $reduce(q, g_1, g_2)$ returns $q(x) \leftarrow P_2(x, y)$ and since now $y$ is unbound, after application of $\kappa$, step 10. inserts the query $q_3$, $q(x) \leftarrow P_2(x, \_)$. At the third execution of step 7., the algorithm inserts query $q_4$, $q(x) \leftarrow B(x)$ using the axiom $B \sqsubseteq \exists P_2$ and stops.*

*We point out that we need not to evaluate all queries. Indeed, it can be verified that for each query $q_i$, all constants $c$ and interpretations $\mathcal{I}$ either ${q_3}^{\mathcal{I}}(c) \geq {q_i}^{\mathcal{I}}(c)$ or ${q_4}^{\mathcal{I}}(c) \geq {q_i}^{\mathcal{I}}(c)$. That is, we can restrict the evaluation of the set of reformulated queries to $\{q_3, q_4\}$ only. As a consequence, the top-2 answers to the original query are the tuples $\langle \mathtt{a}, 1\rangle, \langle \mathtt{e}, 0.9\rangle$, which are the top-2 ranked tuples of the union of the answer sets of $q_3$ and $q_4$.*

The main property of the query reformulation algorithm is as follows. It can be shown that

$$ans_k(\mathcal{K}, q) = \mathtt{Top}_k(\{\langle \mathbf{c}, glb(\mathcal{A}, q_i(\mathbf{c}))\rangle \mid q_i \in r(q, \mathcal{T})\rangle\}) \ .$$

The above equation dictates that the set of reformulated queries $q_i \in r(q, \mathcal{T})$ can be used to find the top-$k$ answers, by evaluating them over the set of instances $\mathcal{A}$ only, without referring to the ontology $\mathcal{T}$ anymore.

In the following, we show how to find the top-$k$ answers of the union of the answer sets of conjunctive queries $q_i \in r(q, \mathcal{T})$.

We first note that each conjunctive query $q_i \in r(q, \mathcal{T})$ can easily be transformed into an SQL query expressed over $\mathtt{DB}(\mathcal{A})$. The transformation is conceptually very simple. The only non-trivial case concerns binary atoms with unbound terms: for any atom in a query $q_i \in r(q, \mathcal{T})$ of the form $P(\_, x)$, we introduce a view predicate that represents the union of $tab_P[2, 3]$ with $tab_{\exists P^-}$, where $tab_P[2, 3]$ indicates projection of $tab_P$ on its second and third column (similarly for $P(x, \_)$). All SQL queries obtained from $P$, together with the views introduced in the transformation can be easily dispatched to an SQL query engine and evaluated over $\mathtt{DB}(\mathcal{A})$.

Then, a possible, naive solution to the fuzzy retrieval problem may be as follows: we compute for all $q_i \in r(q, \mathcal{T})$ the whole answer set $ans(q_i, \mathcal{A}) = \{\langle \mathbf{c}, glb(\mathcal{A}, q_i(\mathbf{c}))\rangle\}$, take the union, $\bigcup_{q_i \in r(q,\mathcal{T})} ans(q_i, \mathcal{A})$, of these answer sets, order it in descending order of truth degree and then we take the top-$k$ tuples.

A major drawback of this solution is the fact that each tuple satisfies a query to a degree and hence for any query $q_i \in r(q, \mathcal{T})$, all tuples are retrieved always. This is in practice *not feasible*, as a there may be millions of tuples in the knowledge base. Restricting the answer set of each query to the tuples with non-zero degree is not satisfactory as well, as still there might be too many tuples in the answer set.

A more effective solution consists in relying on existing top-$k$ queries answering algorithms (see, e.g. [5, 6, 11]), which support efficient evaluations of ranking top-$k$ queries in relational database systems. This gives us immediately a much more efficient method to compute $ans_k(\mathcal{K}, q)$: we compute for all $q_i \in r(q, \mathcal{T})$, the top-$k$ answers $ans_k(\mathcal{A}, q_i)$, using e.g. the system RankSQL [11]. If both $k$ and the number, $n_q = |r(q, \mathcal{T})|$, of reformulated queries is small, say $k \cdot n_q \leq 100$, then we may take the union, $U(q, \mathcal{K}) = \bigcup_{q_i \in r(q,\mathcal{T})} ans_k(\mathcal{A}, q_i)$, of these top-$k$ answer sets, order it in descending order of truth degree and then we take the top-$k$ tuples.

As an alternative, we can avoid to compute the whole union $U(q, \mathcal{K})$, by relying on a variant of the so-called *Threshold Algorithm* (TA) [7]: let us assume that the tuples in the top-$k$ answer set $ans_k(\mathcal{A}, q_i)$ are sorted in decreasing order with respect to the truth degree. Then we process each top-$k$ answer set $ans_k(\mathcal{A}, q_i)$ ($q_i \in r(q, \mathcal{T})$) in parallel and top-down. ($i$) For each tuple $\mathbf{c}$ seen, if its truth degree is one of the $k$ highest we have seen, then remember object $\mathbf{c}$ and its truth degree $t(\mathbf{c})$ (ties are broken arbitrarily, so that only $k$ tuples and their truth degrees need to be remembered at any time). ($ii$) For each answer set $ans_k(\mathcal{A}, q_i)$, let $t_i$ be the truth degree of the last tuple seen in this set. Define the threshold value $\theta$ to be $\max(t_1, ..., t_{n_q})$. As soon as at least $k$ tuples have been seen whose grade is at least equal to $\theta$, then halt (indeed,

any successive retrieved tuple will have truth-degree $\leq \theta$). (*iii*) Let $Y$ be a set containing the $k$ tuples that have been seen with the highest truth degrees. The output is then the graded set $\{\langle \mathbf{c}, t(\mathbf{c})\rangle \mid \mathbf{c} \in Y\}$. This set is $ans_k(\mathcal{K}, q)$.

The above method, based on existing technology for answering top-$k$ queries over relation databases, improves significantly the naive solution to the fuzzy retrieval problem. Though, it leaves room for improvement. Indeed, we still require to find for each query $q_i \in r(q, \mathcal{T})$ the top-$k$ answer sets $ans_k(\mathcal{A}, q_i)$. An improvement can be obtained by extending current top-$k$ query answering procedures over relational databases, by generalizing them to the retrieval of top-$k$ tuples of the *union* of queries and not just to *one* query. We will deserve to this point more attention in future works.

## 4 Conclusions

Fuzzy DLs have been proposed as a mean to describe structured knowledge with vague concepts and find their natural application in the context of the Semantic Web. A major distinction of fuzzy DLs is that, an answer to a query is a set of tuples *ranked* according to the degree they satisfy the query. As a consequence, whenever we deal with a huge amount of tuples, the ranking of the answer set becomes the major problem that has to be addressed to make fuzzy DLs viable for real-world application.

In this paper we have considered fuzzy DL-Lite and have shown how to answer complex queries (in particular, conjunctive queries) efficiently over a huge set of instances. The main ingredients of our solution is a simple and effective query reformulation procedure and the use of existing top-$k$ query answering technology over relational databases. Indeed, a user query is reformulated into a set of conjunctive queries using the inclusion axioms only and, then, the reformulated queries can be submitted to the top-$k$ query answering engine over a relational database where the tuples have been stored. The proposed solution allows to deal with virtually millions of tuples, depending on the effectiveness of the DBMS.

Related to future research, we envisage two directions: (*i*) to verify the applicability of our method to richer fuzzy DLs than DL-Lite, or other tractable DLs such as $\mathcal{EL}$ [2]; and (*ii*) as, stated at the end of the previous section, to improve the core top-$k$ query answering technology towards the management of the union of queries, starting from the existing literature on top-$k$ query answering procedures.

## References


[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[2] F. Baader, S. Brandt and C. Lutz. Pushing the $\mathcal{EL}$ Envelope. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence IJCAI-05*, 2005.

[3] A. Calì and D. Lembo, and R. Rosati. Query rewriting and answering under constraints in data integration systems. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI-03)*, 2005.

[4] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. DL-Lite: Tractable description logics for ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, 2005.

[5] K. Chang and S. Hwang. Minimal probing: Supporting expensive predicates for top-k queries. In *SIGMOD Conference*, 2002.

[6] R. Fagin. Combining fuzzy information: an overview. *SIGMOD Rec.*, 31(2):109–118, 2002.

[7] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Symposium on Principles of Database Systems*, 2001.

[8] P. Hájek. *Metamathematics of Fuzzy Logic*. Kluwer, 1998.

[9] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[10] H. J. Levesque and G. Lakemeyer. *The Logic of Knowledge Bases*. MIT Press, 2001.

[11] C. Li, K. Chang, I. F. Ilyas, and S. Song. RankSQL: query algebra and optimization for relational top-k queries. In *SIGMOD Conference*, 2005.

[12] John W. Lloyd. Foundations of Logic Programming. Springer Verlag, 1987.

[13] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.

[14] U. Straccia. Reasoning within fuzzy description logics. *Journal of Artificial Intelligence Research*, 14:137–166, 2001.

[15] U. Straccia. Description logics with fuzzy concrete domains. In *21st Conf. on Uncertainty in Artificial Intelligence (UAI-05)*, pages 559–567, 2005. AUAI Press.

[16] U. Straccia. A fuzzy description logic for the semantic web. In Elie Sanchez, editor, *Capturing Intelligence: Fuzzy Logic and the Semantic Web*. Elsevier, 2006. To appear.