FREIE UNIVERSITÄT BOZEN
LIBERA UNIVERSITÀ DI BOLZANO
FREE UNIVERSITY OF BOZEN - BOLZANO

KRDB RESEARCH CENTRE
KNOWLEDGE REPRESENTATION MEETS DATABASES

Faculty of Computer Science, Free University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bolzano, Italy
Tel: +39 04710 16000, fax: +39 04710 16009, http://www.inf.unibz.it/krdb/

KRDB Research Centre Technical Report:

Explanation in *DL-Lite*

Alexander Borgida[1], Diego Calvanese[2], Mariano Rodriguez-Muro[2]

Affiliation	[1] Dept. of Computer Science, Rutgers University, USA [2] Faculty of Computer Science, Free University of Bozen-Bolzano, Italy
Corresponding author	Diego Calvanese calvanese@inf.unibz.it
Keywords	Description Logics, query answering, explanation
Number	KRDB08-2
Date	18-02-2008
URL	http://www.inf.unibz.it/krdb/pub/

Paper submitted to the 2008 International Workshop on Description Logics.

Explanation in *DL-Lite*

Alexander Borgida[1], Diego Calvanese[2], Mariano Rodriguez-Muro[2]

[1] Dept. of Computer Science
Rutgers University, USA

[2] Faculty of Computer Science
Free University of Bozen-Bolzano, Italy

Abstract. The paper addresses the problem of explaining some reasoning tasks associated with the *DL-Lite* Description Logic. Because of the simplicity of the language, standard concept level reasoning is quite easy, and the only contribution is an alternate, more accessible syntax, plus a focus on brevity of proofs. Of greater interest is the explanation of reasoning in finite models, which is motivated by the use of *DL-Lite* for database access. The fame of *DL-Lite* rests on its ability to answer efficiently conjunctive queries over KBs, and the paper makes three contributions in this regard: (1) a method for explaining why a value b was returned by a query; (2) a method for finding minimal explanations for why a conjunctive query is *unsatisfiable*; (3) the beginnings of a theory for explaining why a value b was **not** returned by a query.

1 Introduction

It is by now widely accepted that end-users of information systems which do more than simple fact retrieval require some sort of facility for *explaining* the reason for their answers. For example, in the area of deductive databases there has been work on explaining answers returned by Datalog-query processors [1].In Description Logics (DLs), starting from the work in [2], there have been a number of papers studying the explanation of deductions such as concept subsumption (e.g., [3]) and knowledge base inconsistency (e.g., [4]).More generally, the work on the Inference Web [5] has produced a substrate on which general explanation facilities for reasoners can be built.

DL-Lite is a DL that was introduced to capture most of the features of conceptual modeling languages such as ER and UML, while at the same time maintaining efficient query processing (low data complexity) similar to standard databases [6].

The present research considers the problem of explaining reasoning and query answering for *DL-Lite*. As for any DL, there are standard judgements such as concept subsumption, TBox , and ABox consistency, etc. which require more or less straightforward explanations. Because of its use for database conceptual modeling, and the fact that databases are almost always considered to represent *finite structures* in which the conceptual models are interpreted, one important novel feature of the above reasoning tasks is the possibility of requiring finite models (which for *DL-Lite* enable additional inferences, i.e., the logic lacks the finite-model property!). A second distinguishing feature of *DL-Lite* is the emphasis on *conjunctive query processing*, which will require new kinds of explanations. The rest of the paper has the following structure: Section 2 provides formal background on *DL-Lite* and explanations; Section 3 considers the relatively straightforward reasoning tasks associated with *DL-Lite*, but also looks at the

more unusual notion of finite-model reasoning; Section 4 considers in detail explaining why some value is returned as one of the answers to a conjunctive query posed to a *DL-Lite* ABox; while Section 5 discusses the difficulties of explaining why a particular value was *not* returned in a conjunctive query answer, even in the case of regular databases, and gives a special-purpose solution for the case when this occurs because the query is itself unsatisfiable.

2 Background

For reasons of space, we deal only with the *DL-Lite*$_{\mathcal{F}}$ variant of the *DL-Lite* family of DLs [6]. However, the present work can be extended with little effort also to other variants, so we continue to use the term *DL-Lite*.

Starting from atomic concepts A and atomic roles S, in *DL-Lite* one can construct so called *basic concepts* B of the form A, S, and S^-. In addition, we make use of the negation $\neg B$ of a basic concept B. The semantics is the standard one, i.e., given an interpretation $\mathcal{I} = \ ^{\mathcal{I}}, \cdot^{\mathcal{I}}$, a concept C denotes a subset $C^{\mathcal{I}}$ of the interpretation domain $\ ^{\mathcal{I}}$, and a role S denotes a binary relation $S^{\mathcal{I}} \ \ ^{\mathcal{I}} \times \ ^{\mathcal{I}}$, satisfying $(\ S)^{\mathcal{I}} = \{x \mid \ y.(x,y) \ \ S^{\mathcal{I}}\}$, $(\ S^-)^{\mathcal{I}} = \{y \mid \ x.(x,y) \ \ S^{\mathcal{I}}\}$, and $(\neg B)^{\mathcal{I}} = \ ^{\mathcal{I}} \setminus B^{\mathcal{I}}$.

A knowledge base (KB) $\mathcal{K} = \ \mathcal{T}, \mathcal{A}$ consists of a *TBox* $\mathcal{T}$ of *terminological axioms* and an *ABox* $\mathcal{A}$ of *facts* about individuals. Axioms in $\mathcal{T}$ have the form $B_1 \ \ B_2$, $B_1 \ \ \neg B_2$, and $(\mathsf{funct}\ S)$ and assert, respectively, subsumptions and disjointness between two basic concepts B_1 and B_2, and the functionality of role S. An interpretation $\mathcal{I}$ *satisfies* an assertion $B \ \ C$ if $B^{\mathcal{I}} \ \ C^{\mathcal{I}}$, and it satisfies $(\mathsf{funct}\ S)$ if $\ x, y, z.(x,y) \ \ S^{\mathcal{I}} \ \ (x,z) \ \ S^{\mathcal{I}} \ \ y = z$. Each individual d in $\mathcal{A}$ is interpreted as a domain element $d^{\mathcal{I}} \ \ ^{\mathcal{I}}$ under the unique name assumption. Assertions in $\mathcal{A}$ have the form $A(d)$ and $S(d_1, d_2)$, and they are satisfied in an interpretation $\mathcal{I}$ if $d^{\mathcal{I}} \ \ A^{\mathcal{I}}$ and $(d_1^{\mathcal{I}}, d_2^{\mathcal{I}}) \ \ S^{\mathcal{I}}$, respectively.

An interpretation that satisfies all assertions in $\mathcal{K}$ is called a *model* of $\mathcal{K}$. The reasoning tasks of KB satisfiability, subsumption, classification, and concept consistency are defined in the standard way, by referring to the set of models of a KB. *Finite model reasoning* restricts the attention only to models whose domain is finite.

We are interested also in answering conjunctive queries (CQs) over *DL-Lite* KBs. A CQ over a *DL-Lite* KB $\mathcal{K}$ has the form $Q(\boldsymbol{x}) \ \ conj(\boldsymbol{x}, \boldsymbol{y})$, where $conj(\boldsymbol{x}, \boldsymbol{y})$ is a conjunction of atoms whose predicates are atomic concepts and roles of $\mathcal{K}$, and whose variables are among $\boldsymbol{x}$ and $\boldsymbol{y}$. $Q(\boldsymbol{x})$ is interpreted in $\mathcal{I}$ as the set $Q(\boldsymbol{x})^{\mathcal{I}}$ of tuples $\boldsymbol{d}$ of elements of $\ ^{\mathcal{I}}$ such that, when we assign $\boldsymbol{d}$ to $\boldsymbol{x}$, the formula $\ \boldsymbol{y}.conj(\boldsymbol{x}, \boldsymbol{y})$ evaluates to true in $\mathcal{I}$. The set of *certain answers* to $Q(\boldsymbol{x})$ over $\mathcal{K}$ is the set of tuples $\boldsymbol{c}$ of ABox individuals such that $\boldsymbol{c}^{\mathcal{I}} \ \ Q(\boldsymbol{x})^{\mathcal{I}}$, for every model $\mathcal{I}$ of $\mathcal{K}$. $Q_1(\boldsymbol{x})$ is *consistent* wrt $\mathcal{K}$, if there is a model $\mathcal{I}$ of $\mathcal{K}$ such that $Q(\boldsymbol{x})^{\mathcal{I}} = \ .\ Q_1(\boldsymbol{x})$ is *contained in* $Q_2(\boldsymbol{x})$) wrt $\mathcal{K}$ if $Q_1(\boldsymbol{x})^{\mathcal{I}} \ \ Q_1(\boldsymbol{x})^{\mathcal{I}}$, for every model $\mathcal{I}$ of $\mathcal{K}$. When the ABox $\mathcal{A}$ is treated as a "closed database" (i.e., a single interpretation), $Q(\boldsymbol{x})^{\mathcal{A}}$ denotes the evaluation of $Q(\boldsymbol{x})$ in $\mathcal{A}$.

As far as explanations, it is almost universally accepted that they are formal *proof*s, constructed from premises using rules of inference. For better explanations, proofs should be as short as possible, and inference rules as self-evident as possible. In addition to the proof, there is also the matter of its *presentation*. There is a decided preference (e.g.,[2, 5]) for tree-shaped proofs, produced by rules of inference with a single conclusion, and zero or more antecedents. Such proofs support, through follow-up questions,

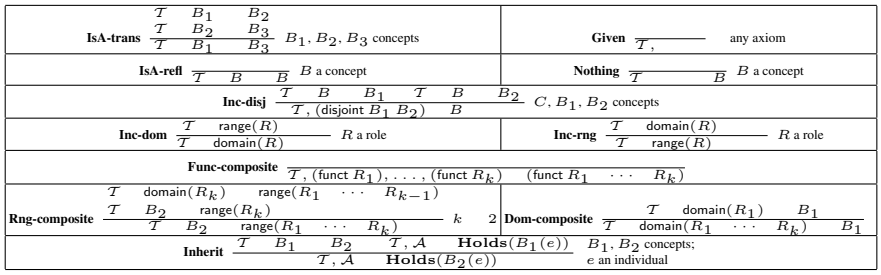

Fig. 1. Inference rules

the interactive and gradual unfolding of relevant parts under user control. Although [5] suggests a specific XML-based syntax for inference rule schemas, we will use the more concise notation used in Programming Languages, and first applied to DLs in [7], but which can easily be translated to the rules in [5].

We use the inference rule **Isa-trans** in Figure 1, expressing the transitivity of the relationship, to illustrate the notation. Here, the name of the rule schema is **Isa-trans**, the antecedent says that $B_1 \quad B_2$ and $B_2 \quad B_3$ can be deduced from $\mathcal{T}$, while the conclusion allows one to also deduce $B_1 \quad B_3$ from $\mathcal{T}$; the side-condition of the rule requires B_1, B_2, and B_3 to be concept expressions.

3 Explanations of Standard Inferences

3.1 Modified Syntax of *DL-Lite*

A number of notions in *DL-Lite* (and their notation), such as existential constraints, inverses of roles, and complements of concepts are mathematically too sophisticated for users familiar only with notations like UML diagrams. For this reason, we propose to eliminate the notations S and S^-, and replace them by the more familiar notions of "the current domain of role S" and "the current range of role S", written as $\mathsf{domain}(S)$ and $\mathsf{range}(S)$[1].

The only remaining use of role inverses is in axioms of the form $(\mathsf{funct}\ S^-)$, as in $(\mathsf{funct}\ makes^-)$. Normally, we would suggest replacing $makes^-$ by$madeBy$, and $\mathsf{domain}(makes)$ (resp., $\mathsf{range}(makes)$) by $\mathsf{range}(madeBy)$ (resp., $\mathsf{domain}(madeBy)$). This leaves the case of having both $(\mathsf{funct}\ S)$ and $(\mathsf{funct}\ S^-)$ in the TBox (e.g., $(\mathsf{funct}\ wife)$,$(\mathsf{funct}\ wife^-)$). To deal with these, while eliminating the S^- notation, we permit declaring a name for the inverse role, using an axiom of the form $(\mathsf{inverseFunctionalRoles}\ wifeOf\ husbandOf)$, which introduces pairs of inverse roles but also requires these to be functions. To eliminate $\neg$, we also replace each axiom of the form $B_1 \quad \neg B_2$, expressing disjointness of two basic concepts, by the axiom $(\mathsf{disjoint}\ B_1\ B_2)$, having the same semantics.

As a result of the above simplifications, subsumption axioms will now only relate atomic concepts and/or current domains/ranges of roles. And in addition to subsumption, we have axioms for disjointness of concepts, functionality of roles, and declarations of names for role inverses.

[1] The use of the word *"current"* is meant to emphasize the distinction from OWL "domain", which describe the *potential* set of objects to which a property may apply.

3.2 TBox Reasoning

Subsumption reasoning in *DL-Lite* is a particularly simple form of structural subsumption, in part because there are no nested concepts. Therefore one does not need any of the complications suggested in [2], such as atomic concepts etc., and for normalized concepts, the standard -inference rules for givens, reflexivity and transitivity suffice. Moreover, a *DL-Lite* concept can only be unsatisfiable either due to subsumption by disjoint concepts, or because it is subsumed by the current domain (resp., range) of a role whose current range (resp., domain) is unsatisfiable. Hence, we get (see Figure 1)

Proposition 1 *For DL-Lite, the following is a sound and complete set of rules of inference:* {**Given**, **IsA-refl**, **IsA-trans**, **Nothing**, **Inc-disj**, **Inc-dom**, **Inc-rng**}.

This means that all reasoning about reduces to simple classification of atomic concepts and expressions denoting the current domains/ranges of roles, i.e., computing the so-called Hasse diagram induced by the axioms in the TBox, plus detecting inconsistency. Once this is done, explaining B_1 B_2 for satisfiable concepts involves only finding the shortest path between them.

To explain unsatisfiability of a concept B, we have to find two -paths from B to two concepts, say B_1, B_2, asserted to be disjoint, whose total length is minimal[2]. To find this, one can use a breadth-first-search strategy, where rule **Isa-trans** is applied in parallel waves, and each concept inferred to subsume B is decorated with a superscript that is one higher than that of the antecedent concept, thereby capturing the wave when it was added. (B is assigned superscript 0.) Then, if a disjointness is detected the first time between ${}^{n}B_1$ and ${}^{j}B_2$ (with n j, say), then the length of the explanation for this is $n + j$. Unfortunately, this may not be the shortest explanation, because the total length of such an explanation could be as high as $2n$; on the other hand, there might be another disjointness between, say, ${}^{1}B_3$ and ${}^{n+1}B_4$ added in the $(n + 1)$st pass say, which would only have total explanation length $n + 2$. To find the shortest explanation one must therefore continue beyond wave n up to wave $2n - 1$.

3.3 ABox Reasoning

In *DL-Lite*, one infers new facts about existing individuals by applying inclusion axioms, and recognizing that $R(a, b)$ entails $\mathsf{domain}(R)(a)$ and $\mathsf{range}(R)(b)$.[3]

To detect inconsistencies in a KB, one simply looks for objects belonging to concepts which can be deduced to be subsumed by , or to violated functionality constraints. The one nontrivial aspect is that, as usual, we prefer shorter explanations. In the case of inconsistent ABoxes, one wants *the shortest derivation of a conflict* from the original ABox – one with fewest rule applications. To find this, one can use a strategy similar to the one described above for finding the shortest proof of inconsistency, using rule **Inherit** instead of rule **Isa-trans**.

We note that while the above looks for evidence of knowledge base inconsistency, this is not the same problem as diagnosing errors in the knowledge base. Pinpointing [4], and related orthogonal techniques are much more likely to be useful for this task.

[2] We would have to consider also paths from B to some $\mathsf{domain}(R)$ (resp., $\mathsf{range}(R)$) for which $\mathsf{range}(R)$ (resp., $\mathsf{domain}(R)$) is unsatisfiable, but we simplify here for lack of space.

[3] ABox reasoning can also be formalized with inference rules, using a judgment $\mathbf{Holds}(\ )$ for facts , as in rule **Inherit**, but lack of space prevents us from giving these axioms.

3.4 Reasoning in Finite Models

Example 1. Consider the following TBox

$$(\mathsf{funct}\ tutors) \qquad\qquad Student \qquad \mathsf{range}(tutors)$$
$$\mathsf{domain}(tutors) \qquad TA \qquad\qquad TA \qquad Student$$

Since, $tutors$ is a function, there can be at most as many values in its range as in its domain. Since the current range of $tutors$ contains all $Student$s, there can be at most as many $Student$s as values in the domain of $tutors$. And since $\mathsf{domain}(tutors)$ is contained in the set of TAs, there can be at most as many $Student$s as TAs. If, in addition, now one has that TA $Student$, this implies that there are at most as many TAs as $Student$s, and therefore the number of TAs and $Student$s is the same. In an infinite model, this leads to no new conclusions, even if one recalls that TA is a subset of $Student$. However, in a finite model, these two facts imply that the extensions of TA and $Student$ must be identical, which means that a new subsumption has been inferred: $Student$ TA.

Clearly, the above pattern can be generalized by replacing $tutors$ with the composition of an arbitrary set of roles R_1 R_2 $\cdots$ R_k, obtaining rule **Same-cardinality**:

$$\frac{\mathcal{T} \quad (\mathsf{funct}\ R_1 \quad \cdots \quad R_k) \qquad\qquad \mathcal{T} \quad B_2 \quad \mathsf{range}(R_1 \quad \cdots \quad R_k)}{\mathcal{T} \quad \mathsf{domain}(R_1 \quad \cdots \quad R_k) \quad B_1 \qquad \mathcal{T} \quad B_1 \quad B_2 \qquad\qquad \mathcal{T} \quad B_2 \quad B_1} \qquad \begin{array}{l} B_1, B_2 \text{ concepts;} \\ R_1 \text{ a role} \end{array}$$

The remaining question is how one can deduce properties of a composition of roles, given only *DL-Lite* axioms. First, rule **Func-composite** captures that if all roles are functions, then their composition will be a function. And since the current domain of a composition is contained in the current domain of the first role, we also have rule **Dom-composite**. However, B_2 $\mathsf{range}(R_1 \quad \cdots \quad R_k)$ does not follow from B_2 $\mathsf{range}(R_k)$ alone, because the *current* range of the composition may be smaller, if not all values in $\mathsf{domain}(R_k)$ are reached by R_1 $\cdots$ R_{k-1}. So one also needs the entire current domain of R_k to be contained in the current range of R_1 $\cdots$ R_{k-1}, leading to the rule **Rng-composite**.

It follows from results by [8] that these are *all* the possible additional subsumption inferences needed for the finite model case.

Proposition 2 *The only additional inference rules needed for deducing subsumptions in the case of DL-Lite TBoxes with finite domains are* **Same-cardinality**, **Func-composite**, **Dom-composite**, **Rng-composite**.

As far as explanations are concerned, this is a prime example where the user will need separate explanations for the rules of inference themselves.

4 Explaining positive answers for queries over an ABox

Consider first the simple issue of answering conjunctive queries over a regular database. To explain why $Q(b)$ is true in a database requires showing why the database, treated as an interpretation, makes the body of the query evaluate to true. For conjunctive queries, the only interesting aspect is exhibiting the values used for existentially quantified variables. For example, if `MIMI` is an answer to query $Q_0(x)$

$Student(x)$, $supervisedBy(x, y)$, $teaches(y, z)$, one would locate some "witness" values ANNA and ENG101 for variables y and w, and then explain that $Student(\texttt{MIMI})$, $supervisedBy(\texttt{MIMI}, \texttt{ANNA})$, and $teaches(\texttt{MIMI}, \texttt{ENG101})$ are given in the database.

In the case of ***DL-Lite***, the difficulty is that the ABox is not a closed database, but instead must be "completed" according to the axioms. For example, if we have $Student(\texttt{MIMI})$ and $Student \quad Person$, then we must also add $Person(\texttt{MIMI})$; and if we have $Professor(\texttt{TOM})$ and $Professor \quad \mathsf{domain}(teaches)$, then one can conclude that there is some hypothetical individual, say $@c$, representing what TOM teaches, and that $teaches(\texttt{TOM}, @c)$ holds. Such hypothetical individuals may also get additional properties. Unfortunately, the result can be an infinite database since the axioms may contain cyclic dependencies; e.g., $Nat \quad \mathsf{domain}(succ)$, $\mathsf{range}(succ) \quad Nat$.

From theoretical results [6], we do however know that from the TBox $\mathcal{T}$ and the ABox $\mathcal{A}$ one can derive a (generally infinite) canonical model $can(\mathcal{T}, \mathcal{A})$, representing all possible databases extending $\mathcal{A}$, as far as CQ evaluation is concerned. Essentially, $can(\mathcal{T}, \mathcal{A})$ extends the facts in the ABox as illustrated in the example above. The crucial property of $can(\mathcal{T}, \mathcal{A})$ is that each CQ Q can be answered by evaluating it, as in regular databases, over only a finite "small" portion of $can(\mathcal{T}, \mathcal{A})$ (whose size depends on Q).

We exploit this fact to explain why $Q(b)$ holds, by essentially constructing the *finite part* of $can(\mathcal{T}, \mathcal{A})$ that is needed to explain the truth of the query body for $Q(b)$. In order to generate the relevant part of $can(\mathcal{T}, \mathcal{A})$, we resort to a (suitably adapted version of) the algorithm for query answering in ***DL-Lite***. Query answering in ***DL-Lite*** [6] is performed by first rewriting the original query $Q(x)$ into a set $\mathcal{S} = \{Q_0(x), \ldots, Q_k(x)\}$ of alternate queries, then evaluating these over the original ABox (treated as a closed database), and finally returning the union of the results.

Example 2. Consider the following TBox $\mathcal{T}$

$$PhD \quad Student \tag{1}$$
$$PhD \quad \mathsf{domain}(supervisedBy) \tag{2}$$
$$\mathsf{range}(supervisedBy) \quad Professor \tag{3}$$
$$Professor \quad \mathsf{domain}(teaches) \tag{4}$$

and the query $Q_0(x) \quad Student(x), supervisedBy(x, y), teaches(y, z)$. The ***DL-Lite*** rewriting algorithm would then rewrite Q_0 into the following set of queries:

$$\begin{array}{ll} Q_0(x) & Student(x), supervisedBy(x, y), teaches(y, z) \\ Q_1(x) & PhD(x), supervisedBy(x, y), teaches(y, z) \\ Q_2(x) & PhD(x), supervisedBy(x, y), Professor(y) \\ Q_3(x) & PhD(x), supervisedBy(x, y), supervisedBy(w, y) \\ Q_4(x) & PhD(x), supervisedBy(x, y) \\ Q_5(x) & PhD(x), PhD(x) \end{array}$$

We recall briefly, using the above query as an example, the basic steps of the rewriting algorithm that are necessary to understand its use in our explanation setting; for the full details we refer to [6]. Essentially, the algorithm makes use of replacement and unification steps. A replacement can be applied to an atom when the corresponding predicate appears on the right hand side of an inclusion axiom; e.g., query $Q_1(x)$ is obtained from $Q_0(x)$ by replacing $Student(x)$ with $PhD(x)$, due to axiom (1). Similarly, $Q_2(x)$

is obtained from $Q_1(x)$ by making use of axiom (4).[4] On the other hand, a unification step collapses two atoms with the same predicate when the corresponding variables can be unified; e.g., query $Q_4(x)$ is obtained from $Q_3(x)$ by unifying the two atoms $supervisedBy(x, y)$ and $supervisedBy(w, y)$.

The key observation is that the replacement rewriting steps correspond to the "inverses" of the additions made to the canonical model of the knowledge base. Hence, we can use them to guide the explanation of why a given individual is in the certain answers to the original query. Suppose that an individual b is part of the certain answers to $Q(x)$, and we want to explain why[5]. In our example, suppose the ABox contains $PhD(\texttt{BOB})$, and no other facts about BOB, and we want to explain why BOB is in the answer to $Q_0(x)$. To do so, we proceed as follows.

Step 1. We compute the set $\mathcal{S}$ of rewritings of $Q(x)$, building a data structure that tracks how (changed) atoms in one query are derived from a predecessor query.

In our example, queries $Q_0(x)$ to $Q_5(x)$ will be part of the set $\mathcal{S}$ of rewritings, though if there were other axioms, there may be many more, irrelevant to BOB.

Step 2. Since $Q(b)$ was true, we select from $\mathcal{S}$ a rewriting $Q_k(x)$ that produces b as an answer when directly evaluated over the ABox (viewed as a closed database)[6]. This means that there is an assignment of ABox individuals to the variables in $Q_k(x)$ such that $(x) = b$, and for each atom in $Q_k(x)$, ( ) is an ABox fact.

In our example, we would select $Q_5(x)$, with $(x) = \texttt{BOB}$, where $PhD(\texttt{BOB})$ in the ABox would satisfy the two identical atoms of $Q_5(x)$.

Step 3. Traversing backwards the sequence of rewritings from $Q_k(x)$ to $Q(x)$, we extend the substitution to the variables in only the intervening queries, by keeping track of unifications, or assigning to such variables newly introduced *skolem constants* (corresponding to objects introduced by the inclusion axioms). More precisely, when going backwards from Q_i to Q_j (the query from which Q_i was generated), if no variable has been eliminated when rewriting Q_j to Q_i, then need not be extended. When a variable z has been eliminated because it has been unified with a variable y, then we set $(z) = (y)$. While when a variable z has been eliminated in Q_i by replacing an atom $A(y)$ for $R(y, z)$ (resp., $R(z, y)$), due to inclusion axiom A $\mathsf{domain}(R)$ (resp., A $\mathsf{range}(R)$), then we set $(z) = @c$, where $@c$ is a fresh skolem constant. In this way, when one reaches the original query $Q(x)$, will have assigned to each variable appearing in it either an ABox individual or a skolem constant. In analogy to the case of standard databases, one then initially shows to the user $(Q(x))$, i.e., the set of atoms of the original query to which has been applied.

In our example, we would get $(x) = \texttt{BOB}$ for $Q_5(x)$, and the following new assignments: $(y) = @1$ for $Q_4(x)$, $(w) = (x) = \texttt{BOB}$ for $Q_3(x)$, and $(z) = @2$ for $Q_1(x)$. The resulting initial explanation shown to the user would be the sequence of ground atoms matching the query conjuncts:

$$Q_0(\texttt{BOB}) \qquad Student(\texttt{BOB}), supervisedBy(\texttt{BOB}, @1), teaches(@1, @2)$$

[4] Technical note: a replacement where a variable (z, in the example) is removed from the resulting query is allowed only when such a variable does not occur anywhere else in the query.

[5] Everything below can be generalized for the case when the query returns a tuple of values.

[6] By completeness of the query answering algorithm based on query rewriting [6], such a rewriting $Q_k(x)$ will always exist.

Step 4. At this point, the user is allowed to ask follow-up explanations for any of the atoms $f = (\ )$ appearing above. The data structure set up in Step 1 allows one to find either that f is given in the ABox, or the first replacement (not unification!) which replaced by say. The explanation for f is then $(\ )$ together with the inclusion axiom supporting it. In our example, if the user asks for the explanation of $Student(\texttt{BOB})$, the answer is "$PhD(\texttt{BOB})$, because PhDs are $Students$." obtained from the Q_0 to Q_1 rewriting, based on axiom (1). And the explanation of $teaches(@1, @2)$, is "$Professor(\texttt{BOB})$, because $Professor$s are in the domain of $teaches$" obtained from the Q_1 to Q_2 rewriting, based on axiom (4).

A critical point in the above explanation procedure is the selection of the rewriting $Q_k(x)$ in Step 2. First, we need an efficient way of selecting only the indexes k of those rewritings that actually contribute to $Q(b)$. This can be done by associating to each of the queries $Q_i(x)$ in $\mathcal{S}$ a distinct tag, and returning such tags as part of the answer. If $\mathcal{S} = \{Q_1(x), \ldots, Q_n(x)\}$, then the modified rewriting is written in SQL as:

```
SELECT x, 1 AS tag FROM ... WHERE Q1(x)  UNION
SELECT x, 2 AS tag FROM ... WHERE Q2(x)  UNION ...
```

If more than one query in $\mathcal{S}$ returns b as an answer, it might be best to explain the one whose total number of atoms *plus* number of applied transformations is minimal. (This assumes that users find explanations of ground atom look-ups as easy to understand as axiom applications. If not, one can weight the latter more heavily.)

5 Failed Answers to Conjunctive Queries — First Steps

The previous section dealt with the problem of explaining why some individual b was returned by a query $Q(x)$, i.e., why $Q(b)$ was "satisfied" in the current KB. In this section we begin to consider explaining why an individual b was *not* returned by the query $Q(b)$; such explanations are needed because, presumably, the user was expecting $Q(b)$ to be satisfied in the current KB, but in fact it was not.

There are actually two alternative reasons for this: the query itself is unsatisfiable (so it can never return any answers), or just that b was not one of the values returned.

5.1 Conjunctive query unsatisfiability in *DL-Lite*

The reason to treat this case separately is because it is hard to see why one would want to pose a query that would never be able to return an answer — a question akin to why would one want to define an unsatisfiable concept in an ontology.

In the case of ordinary CQs over regular databases, every query is satisfiable, so this problem does not arise. However, in *DL-Lite*, this no longer holds because of the presence of integrity constraints in the TBox. For example, a *DL-Lite* query would be unsatisfiable if it had a sub-query $B_1(x), B_2(x)$ where B_1 and B_2 are disjoint concepts.

In general, one way to determine whether a query $Q(x)$ is unsatisfiable wrt a TBox $\mathcal{T}$ is to check if it is contained wrt $\mathcal{T}$ in the query $Q \ (x) \qquad (x)$. Following [9], we can show that, in general, containment of a query $Q_1(x)$ in a query $Q_2(x)$ wrt $\mathcal{T}$ can be checked by adapting the standard technique based on reducing query containment to query evaluation, i.e.: (*i*) The atoms in the body of $Q_1(x)$ are viewed as a collection of facts in an ABox $\mathcal{A}_{Q_1}$, except that the query variables (e.g., x, y) are replaced by constant (e.g., c_x, c_y) that do *not* have the unique name assumption. (*ii*) Functionality

is propagated in $\mathcal{A}_{Q_1}$, i.e., if $R(a,b)$ and $R(a,b\ )$ are encountered, (funct R) $\mathcal{T}$, and either of b or b (say b) comes from a variable name, then b is unified with b . Let $\mathcal{A}_{Q_1}$ be the resulting ABox. (*iii*) $Q_2(x)$ is evaluated over $\mathcal{T}, \mathcal{A}_{Q_1}$. Then, $Q_1(x)$ is contained in $Q_2(x)$ wrt $\mathcal{T}$ iff c_x is returned as answer in step (*iii*). Now, when $Q_2(x)$ is (x), the only way it can return c_x is when $\mathcal{A}_{Q_1}$ is inconsistent. The explanation of such inconsistencies was discussed in Section 3, though it needs to be enriched with the explanation of the propagation of functionality.

5.2 Explaining failed answers in ordinary databases

Considering the case of CQs over regular databases[7], we are interested in the reason why some value b was *not* returned as part of the answer to query $Q(x)$, e.g.,

$$Q_1(x) \qquad \mathit{Italian}(x), \mathit{friendOf}(x,y), \mathit{Woman}(y), \mathit{drives}(y,z), \mathit{Ferrari}(z)$$

This corresponds to explaining why the body of the query $Q(b)$ evaluates to false in the interpretation consisting of the database, e.g., why $\neg\ y,z.\mathit{Italian}(b)\quad \mathit{friendOf}(b,y)$ $\cdots$ $\mathit{Ferrari}(z)$ evaluates to false. Unfortunately, attempting a direct, value-level explanation of this would likely be impractical because y and z are actually universally quantified, and one would need to exhibit all combinations of values for them in the database. Note however that one kind of failed answer that can be explained trivially is the non-membership of a (set of) individuals in a table: there is nothing to do (except maybe let the user see the values that are in the table).

To the best of our knowledge, the problem of explaining negated queries was addressed only in [10], which considers Datalog queries with negation. Unfortunately, the solution proposed there takes a rule such as $p(x) \qquad t(x,y), s(y,z)$, and explains "Why not $p(2)$?" with the statement "because $t(2,4)$ but not $s(4,Z)$". But this deals with only *one* possible value of y (namely 4), while one actually needs to explain why *none* of the possible values work. Therefore, we devote the rest of this section to uncovering intuitions about appropriate explanations, formalizing these, and then sketching some special cases in which the intuitions can be realized.

So consider query $Q_1(x)$ above. Suppose that `TED` was not returned as one of the answers, i.e., $Q_1(\texttt{TED})^{\mathcal{A}}$ was false. First, note that the simplest explanation for this would be the fact that `TED` is not Italian *or* has no friends at all, i.e., `TED` is not in the current domain of $\mathit{friendsOf}$. (A simple way to handle the second case would be to expand the query so that every time it contains an atom like $\mathit{friendOf}(x,y)$, an additional conjunct of the form $\mathsf{domain}(\mathit{friendOf})(x)$ is added.) Second, an explanation like "`TED` had no female friends who drive a Ferrari" would be considered inappropriate by natural-language speakers if, in fact, it was the case that `TED` had no friends at all, or even if `TED` had no female friends, because the first sentence *presupposes* the existence of objects of the other kind. Third, even if the explanation "`TED` was Italian but had no friends" broke no presuppositions, it would be considered inferior to one which simply stated that "`TED` had no friends", since the second one is more concise. Fourth, after an explanation such as "`TED` had no women friends who drive" (with no failed presuppositions) one should allow explanations for several follow-up questions, especially "Who are all the women friends of `TED`?", with possible follow up questions of

[7] We view the ABox as the database.

why any are present in this set; or, for any object h in the domain of $drives$, "Why isn't h one of TED's women friends?". We generalize, suggesting the following definition:

Definition 1. *The explanation of why $Q(b)^{\mathcal{A}}$ failed for a satisfiable conjunctive query $Q(x)$ is either a single unsatisfied atom in $Q(b)$ (a trivial base case to explain) or starts with finding a sub-query $subq_Q(b, y)$ of $Q(b)$ plus an additional atom $atom_Q(y)$ in $Q(b)$ such that: (*i*) $S := subq_Q(b, y)^{\mathcal{A}}$ is non-empty (to avoid failed presuppositions). (*ii*) None of the elements d in S satisfy $atom_Q$, i.e., are in the table for $atom_Q$ (to ensure that $subq_Q(b, y)$ $atom_Q(y)$ is also a failing sub-query of $Q(b)$). (*iii*) $subq_Q(b, y)$ is minimal, in the sense that no atoms can be removed from it while preserving (i) and (ii).*

Since for any $Q(b)$, there may be more than one such pair of formulas $(subq_Q, atom_Q)$, there are additional possibilities for minimizing explanations, including: (*i*) choose the $subq_Q$ with smallest size, assuming that users rarely want to consider the exact values in S; (*ii*) minimize the size of set S, so there are fewer individuals to be displayed when follow-up questions arise; (*iii*) minimize the total length of follow-up explanations of why any value is/isn't in S. Let us focus here on the first criterion only.

Given conjunctive query Q, one can first construct a directed graph G_Q that has as vertices the variables and constants of Q, and edges corresponding to the atoms in Q. Unfortunately, in general, it is reasonable to expect that minimizing the size of $subq_Q$ will be a combinatorially difficult problem (in the length of the query) because it involves, in principle, considering all subgraphs of G_Q. (A formal proof of the intractability of the problem is deferred to another paper.)

One special case where the problem is tractable occurs for what we call "path queries": ones whose role-induced edges form a single chain in the graph, starting from the constant b (whose absence is to be explained), and where every node has a small, bounded number of associated concepts[8]. Query $Q_1(x)$ is an example of such a path query. In this case, one can start from the node for b, and consider successively longer paths, performing for each the relational join of the tables named by the edges. The first path T such that does not return the empty set, but T does, provides a minimal size $subq_Q$, equal to .

This approach can be generalized to "tree queries" by performing a breadth-first search through the paths of the query graph that start from the root. And in fact one can try to find spanning trees whose join returns the empty set even for arbitrary graphs (though finding the minimal one is likely to remain a hard problem). The case where no sub-query $subq_Q$ is a tree remains an even more open problem because we have not yet found a strategy for presenting $subq_Q$, and follow-up queries, to the user.

5.3 Generalizing to failed answers in *DL-Lite* ABoxes

Even assuming some solution to the problem of explaining non-answers to CQs, generalizing this to the case of *DL-Lite* is a non-trivial task.

In particular, if, as usual in *DL-Lite*, the original query $Q(x)$ is expanded to the set of queries $\mathcal{S} = \{Q_1(x), Q_2(x), \ldots\}$, then in order to explain why b is not in the certain answers to $Q(x)$, it is not sufficient to explain why each $Q_i(b)$ fails over the ABox, for $i = 1, 2, \ldots$; one must also be prepared to explain why there are no additional queries in the set $\mathcal{S}$ — a highly non-trivial task akin to the proof in [6].

[8] This number is bounded in any case by the cardinality of the TBox alphabet.

A better approach will be to once again start from the potentially infinite canonical database, introducing anonymous constants $@c$, etc., except that in this case we must be prepared to answer follow-up questions about why some constants *do not* necessarily have to exist, or why they need not necessarily have some properties. One does this by appealing to statements such as "the only way to require that b has property A is to have one of the concepts currently describing b entail (be subsumed by) A; this is not the case. Would you like to see why?" — a follow-up question which may lead to non-subsumption explanations.

6 Conclusions

We have tackled the problem of explaining *DL-Lite* reasoning, which we view as (*i*) requiring inference rules for building proofs, and (*ii*) finding short proofs. For standard concept level reasoning, essentially we rely on an alternate more accessible syntax. Of greater novelty and complexity is the explanation of reasoning in finite models. Since *DL-Lite* is intended to support efficient CQ evaluation over a DL KB, we address for the first time explanation for this, providing three contributions: (*i*) a theoretically sound technique for explaining why a value was returned by a query; (*ii*) explanation of why a CQ is *unsatisfiable*, which rests on an algorithm for finding minimal length proofs of the unsatisfiability of an ABox; (*iii*) explanation of why a value was **not** returned by a CQ, a problem that, surprisingly, has not been addressed even for CQs over ordinary databases. For lack of space we do not provide inference rules for all aspects, which would make clear how to implement our explanations following [5].

References

1. Shmueli, O., Tsur, S.: Logical diagnosis of LDL programs. In: Proc. of ICLP'90. (1990) 112–129
2. McGuinness, D.L., Borgida, A.: Explaining subsumption in description logics. In: Proc. of IJCAI'95. (1995) 816–821
3. Deng, X., Haarslev, V., Shiri, N.: A framework for explaining reasoning in description logics. In: Working Notes of the AAAI Fall Symposium on "Explanation-aware Computing". (2005) 189–204
4. Schlobach, S.: Debugging and semantic clarification by pinpointing. In: Proc. of ESWC 2005. (2005) 226–240
5. McGuinness, D.L., Pinheiro da Silva, P.: Explaining answers from the Semantic Web: the Inference Web approach. J. of Web Semantics **1** (2004) 397–413
6. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. J. of Automated Reasoning **39** (2007) 385–429
7. Borgida, A.: From type systems to knowledge representation: Natural semantics specifications for description logics. J. of Intelligent and Cooperative Information Systems **1** (1992) 93–126
8. Rosati, R.: Personal comunication (2008)
9. Rosati, R.: On the decidability and finite controllability of query processing in databases with incomplete information. In: Proc. of PODS 2006. (2006) 356–365
10. Specht, G.: Generating explanation trees even for negations in deductive database systems. In: Proc. of the 5th Workshop on Logic Programming Environments (LPE'93). (1993) 8–13