Explanation in the *DL-Lite* Family of Description Logics

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

[1] Dept. of Computer Science
Rutgers University, USA
borgida@cs.rutgers.edu
[2] Faculty of Computer Science
Free University of Bozen-Bolzano, Italy
{calvanese,rodriguez}@inf.unibz.it

Abstract. In ontology-based data access (OBDA), access to (multiple) incomplete data sources is mediated by a conceptual layer constituted by an ontology. In such a setting, to correctly compute answers to queries, it is necessary to perform complex reasoning over the constraints expressed by the ontology. We consider the case of ontologies expressed in *DL-Lite*, a family of DLs that, in the context of OBDA, provide an optimal tradeoff between expressive power and computational complexity of reasoning; notably conjunctive query answering is LOGSPACE in the size of the data. However, query answering with reasoning comes at a price: the justification of the presence of tuples in answers is no longer trivial, and requires *explanation*. In this paper, we characterize reasoning in *DL-Lite*, through deduction rules for building proofs, and we provide several novel contributions: (*i*) For standard ontology level reasoning, explanation is relatively simple, and our contribution comes mainly from a novel focus on brevity of proofs. (*ii*) Motivated by the use of *DL-Lite* for OBDA, we analyze and provide explanation for reasoning in finite models. (*iii*) We provide a facility for the explanation of an answer to a conjunctive query over a *DL-Lite* ontology. This algorithm is able to exploit the relational query engine to extract from the data the information necessary for finding the explanation more efficiently, and thus scales to large data sets. The presented approach has been implemented in a prototype for constructing explanations. [1]

1 Introduction

Semantic data models such as the Extended ER model (EER) and UML are well known to provide a view of an application domain that is closer to the users' conceptualization of it than standard databases. As a result, there have long been proposals for querying databases through interfaces that offer such conceptual schemas to users (e.g., [2]).

On the other hand, Description Logics (DLs) [3] are a family of knowledge representation schemes developed over the past three decades that deal with concepts (unary relationships) and roles (binary relationships), which can be built up from atomic symbols using special concept and role *constructors*. These logics have precise formal semantics, and support sound and complete reasoning about judgments such as whether one concept subsumes/is more general than another, or whether a concept is unsatisfiable.

[1] Preliminary results on the research reported in this paper appeared in the Working Notes of the 2008 Workshop on Description Logics [1].

R. Meersman and Z. Tari (Eds.): OTM 2008, Part II, LNCS 5332, pp. 1440–1457, 2008.

One application of DLs is providing the formal foundation of web ontology languages such as OWL-DL[2]. More relevantly to this paper, it is known that EER and UML conceptual models can be translated into sets of DL axioms (called "TBoxes"), as in [4,5]. As a result, it is possible to use DL reasoners to detect inconsistencies in EER and UML models (e.g., classes that cannot possibly have any instances).

However, it is also well known that more expressive DLs (those with more constructors) tend to have a higher complexity of reasoning. The DL *DL-Lite* [6] was introduced to capture as much as possible of EER and UML conceptual models while still having effective reasoning. Moreover, in various contexts, such as data integration and ontology-based data access [7], data sources can be queried through a conceptual schema (or an ontology) that provides a formalization of the domain of interest. The key difficulty in such a setting is that the data stored in the sources is in general incomplete w.r.t. the constraints imposed by the schema, and hence have to be considered under the "open world assumption". As a consequence, sophisticated reasoning may be required to obtain answers. For example, if the schema specifies that $Undergrads$ are $Students$, and $Students$ must be enrolled in at least two courses, then, in answering the query $q(x) \leftarrow Undergrads(x) \wedge enrolledIn(x, y)$, a system should be able to infer that all instances of $Undergrads$ should be returned, without checking for each of it if there is an *explicitly* mentioned course it is enrolled in. Nevertheless, when the schema is expressed in *DL-Lite*, conjunctive queries can be answered with low data complexity (LOGSPACE, as in ordinary databases), while fully taking into account the constraints imposed by the schema [6].

Reasoning comes however at a price: end-users of information systems that do more than simple fact retrieval require some sort of facility for having answers *explained* to them. For example, in the area of deductive databases there has been work on explaining answers returned by Datalog-query processors [8,9]. Finding such explanations is nontrivial since the performance systems that do query answering are optimized, and do not use straightforward inference rules, such as back-chaining.

In the field of DLs, starting from [10], there have been papers studying the explanation of deductions such as concept subsumption [11,12] and knowledge base inconsistency [13,14,15,16]. More generally, the work on the Inference Web [17] has produced a substrate on which general explanation facilities for reasoners can be built.

Here we consider the problem of explaining reasoning and query answering for *DL-Lite*$_{\mathcal{A}}$, the most expressive variant of the *DL-Lite* family considered in [6]. As for any DL, there are standard judgments such as concept/role subsumption, concept/role satisfiability and consistency of an ontology, requiring more or less standard 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 because the logic lacks the finite-model property). Example 3 in Section 3.4 illustrates first in English and then using inference rules the kind of reasoning that is needed in finite models. A second distinguishing feature of *DL-Lite* is the emphasis on *conjunctive query answering*, which requires new kinds of explanations. Considering the query in

[2] `http://www.w3.org/2007/OWL/`

Algorithm: breadth-first search for finding disjoint ancestors.
Input: concept B; set $\mathcal{T}$ of disjointness and acyclic concept inclusion assertions.
Output: all pairs of concepts X, Y that are $\sqsubseteq$-ancestors of B and are declared disjoint in $\mathcal{T}$.
/* The search is breadth-first in the sense that, if (X_1, Y_1) and (X_2, Y_2) are two output pairs, and $\max(\text{dist}(B, X_1), \text{dist}(B, Y_1)) < \max(\text{dist}(B, X_2), \text{dist}(B, Y_2))$, then (X_1, Y_1) is output first. */
/* Data structures: queues q_1 and q_2 hold pairs (V, k), where V is a node and k is an integer representing the distance from B to V. */

```
{
  new(q1);
  q1.enter(B, 0);    /* start outer BFS from node B */
  while (not q1.empty()) {
    (X, n) ← q1.leave();
    for all D in parents(X, T) { q1.enter(D, n+1); }
    new(q2);
    q2.enter(B, 0);   /* start new BFS from node B */
    while (not q2.empty()) {
      (Y, m) ← q2.leave();
      if (m > n) then exit loop  /* to look at smallest m+n pairs only */
      for all D in parents(Y, T) { q2.enter(D, m+1); }
      if disjoint(X, Y, T) then
        print X + ", " + Y + " are a source of unsatisfiability at proof length " + (n+m);
    }
  }
}
```

Fig. 1. Breadth-first search algorithm for finding disjoint ancestors

Student(BOB)
  { by **Subconcept** rule from
    $PhD \sqsubseteq Student$ { Axiom 1 }
    PhD(BOB) { DB fact } }
supervisedBy(BOB, @1) (*)
  { by **Subconcept** rule from
    $PhD \sqsubseteq \mathsf{dom}(supervisedBy)$ { Axiom 2 }
    PhD(BOB) { DB fact } }
teaches(@1, @2)
  { by **Subconcept** rule from
    $Professor \sqsubseteq \mathsf{dom}(teaches)$ { Axiom 4 }
    Professor(@1)
      { by **Subconcept** rule from
        $\mathsf{rng}(supervisedBy) \sqsubseteq Professor$ { Axiom 3 }
        supervisedBy(BOB, @1) { see (*) } } }

Fig. 2. Proof tree generated from Step 4 for Q_5(BOB)

Example 4, the kind of explanation needed in this case for the answer BOB is shown in Figure 2.

The rest of the paper has the following structure: Section 2 provides formal background on *DL-Lite* and general desiderata for explanations; Section 3 considers the

relatively straightforward reasoning tasks associated with *DL-Lite*, characterizing them in terms of inference rules, but also looks at the more unusual notion of finite-model reasoning; Section 4 considers in detail the task of explaining why some value is returned as one of the answers to a conjunctive query posed to a *DL-Lite* ontology.

2 Background

In this section we provide the formal background for the techniques and results in the rest of the paper. Specifically, we first introduce the Description Logic we deal with, and then provide some basic notions about explanations.

2.1 The *DL-Lite* Family of Description Logics

Description Logics (DLs) [3] are logics that represent the domain of interest in terms of *concepts*, denoting sets of objects, and *roles*, denoting binary relations between (instances of) concepts. Complex concept and role expressions are constructed starting from a set of atomic concepts and roles by applying suitable constructs, that depend on the DL at hand. In this paper, we deal with the *DL-Lite* family [6,18], which comprises tractable DLs particularly suited for accessing through an ontology large amounts of data managed through relational database technology. Specifically, we consider *DL-Lite*$_{\mathcal{A}}$ [19], one of the most expressive members of the family still enjoying LOGSPACE data complexity of query answering[3].

Concepts and roles in *DL-Lite*$_{\mathcal{A}}$ are formed according to the following syntax:

$$B \longrightarrow A \mid \exists Q \qquad\qquad Q \longrightarrow P \mid P^-$$
$$C \longrightarrow B \mid \neg B \qquad\qquad R \longrightarrow Q \mid \neg Q$$

where A, B, and C respectively denote an *atomic concept*, a *basic concept*, and a *general concept* (or simply, concept), whereas P, Q, and R respectively denote an *atomic role*, a *basic role*, and a *general role* (or simply, role).

Intuitively, a basic role of the form P^- denotes the *inverse* of the relation denoted by role P. A basic concept of the form $\exists P$ (resp., $\exists P^-$) denotes the projection of the relation denoted by P on its first (resp., second) component. An arbitrary concept $\neg B$ (resp., an arbitrary role $\neg Q$) denotes the complement of B (resp., Q).

A DL *knowledge base* (KB) $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ represents the domain of interest and consists of two parts, a TBox $\mathcal{T}$, representing intensional knowledge, and an ABox $\mathcal{A}$, representing extensional knowledge. In *DL-Lite*$_{\mathcal{A}}$, a *TBox* is formed by a set of assertions of the following forms:

$B \sqsubseteq C$	*concept inclusion assertion*
$Q \sqsubseteq R$	*role inclusion assertion*
$(\mathsf{funct}\ Q)$	*functionality assertion*

[3] We ignore here the distinction, present in *DL-Lite*$_{\mathcal{A}}$, between abstract objects and data values.

The concept inclusion assertion $B \sqsubseteq C$ expresses that all instances of the (basic) concept B are also instances of the (general) concept C. Analogously for a role inclusion assertion. A functionality assertion expresses the (global) functionality of a basic role[4].

Example 1. The following TBox

$$PhD \sqsubseteq Student \quad (1) \qquad \exists takes^- \sqsubseteq Course \quad (3)$$
$$\exists takes \sqsubseteq Student \quad (2) \qquad audits \sqsubseteq (\neg takes) \quad (4)$$

asserts that PhDs are a subclass of $Student$s, who are the only ones who can $take$ things, while things taken must be $Course$s; the last axiom is used to express that $takes$ and $audits$ are disjoint. ∎

An *ABox* is formed by a set of *membership assertions* on atomic concepts and on atomic roles of the form

$$A(d) \qquad P(d_1, d_2)$$

stating respectively that the object (denoted by the constant) d is an instance of A, and that the pair (d_1, d_2) of objects is an instance of the role P.

Formally, the semantics of a DL is given in terms of interpretations, where an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of an interpretation domain $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$ that assigns to each concept C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and to each role R a binary relation over $\Delta^{\mathcal{I}}$. In particular, for the constructs of *DL-Lite*$_{\mathcal{A}}$ we have:

$$\begin{array}{rcl@{\qquad}rcl} A^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}} & P^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \\ (\exists Q)^{\mathcal{I}} & = & \{o \mid \exists o' . (o, o') \in Q^{\mathcal{I}}\} & (P^-)^{\mathcal{I}} & = & \{(o_2, o_1) \mid (o_1, o_2) \in P^{\mathcal{I}}\} \\ (\neg B)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus B^{\mathcal{I}} & (\neg Q)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \setminus Q^{\mathcal{I}} \end{array}$$

An interpretation $\mathcal{I}$ *satisfies* an inclusion assertion $B \sqsubseteq C$ (resp., $Q \sqsubseteq R$) if $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$ (resp., $Q^{\mathcal{I}} \subseteq R^{\mathcal{I}}$). Furthermore, $\mathcal{I}$ satisfies an assertion $(\mathsf{funct}\ Q)$ if the binary relation $Q^{\mathcal{I}}$ is a function, i.e., $(o, o_1) \in Q^{\mathcal{I}}$ and $(o, o_2) \in P^{\mathcal{I}}$ implies $o_1 = o_2$. To specify the semantics of membership assertions, we extend the interpretation function to constants, by assigning to each constant a a *distinct* object $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. Note that this implies that, as usual in DLs, we enforce the *unique name assumption* on constants [3]. An interpretation $\mathcal{I}$ satisfies a membership assertion $A(d)$ (resp., $P(d_1, d_2)$) if $d^{\mathcal{I}} \in A^{\mathcal{I}}$ (resp., $(d_1^{\mathcal{I}}, d_2^{\mathcal{I}}) \in P^{\mathcal{I}}$). A *model of a KB* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is an interpretation that satisfies all assertions in $\mathcal{T}$ and $\mathcal{A}$. A KB is *satisfiable* if it has at least one model. A KB $\mathcal{K}$ *logically implies* (an assertion) α, written $\mathcal{K} \models \alpha$, if all models of $\mathcal{K}$ satisfy α. Specifically, a concept C_1 (resp., role R_1) is *subsumed by* a concept C_2 (resp., role R_2) w.r.t. $\mathcal{K}$ if $\mathcal{K} \models C_1 \sqsubseteq C_2$ (resp., $\mathcal{K} \models R_1 \sqsubseteq R_2$). A concept C (resp., role R) is *satisfiable* w.r.t. $\mathcal{K}$ if there is a model $\mathcal{I}$ of $\mathcal{K}$ such that $C^{\mathcal{I}} \neq \emptyset$ (resp, $R^{\mathcal{I}} \neq \emptyset$). Satisfiability and subsumption are the fundamental reasoning tasks over a TBox. Unsatisfiable concepts are typically the result of modeling errors, and should be removed from a KB. Subsumption is the basis of classification, making the structure of the modeled knowledge explicit.

[4] In order to guarantee the computational properties that allow for dealing efficiently with large amounts of data, *DL-Lite*$_{\mathcal{A}}$ requires that, roughly, functional roles cannot be specialized in TBoxes [18,19].

In *DL-Lite*$_\mathcal{A}$, due to the interaction of inclusion and functionality assertions, there may be inferences that do not hold in arbitrary models, but that do hold when only models with a finite domain are considered. In other words, reasoning w.r.t. arbitrary models differs from *finite model reasoning*. All the above reasoning tasks can be defined for the latter case as well.

We are also interested in query answering over *DL-Lite*$_\mathcal{A}$ KBs, and specifically in answering conjunctive queries. A *conjunctive query* (CQ) over a *DL-Lite*$_\mathcal{A}$ KB $\mathcal{K}$ has the form

$$Q(\boldsymbol{x}) \leftarrow conj(\boldsymbol{x}, \boldsymbol{y})$$

where $conj(\boldsymbol{x}, \boldsymbol{y})$ is a conjunction of atoms of the form $A(z)$ or $P(z_1, z_2)$, with A and P respectively atomic concepts and roles of $\mathcal{K}$, and z, z_1, z_2 either constants in $\mathcal{K}$ or variables in $\boldsymbol{x}$ or $\boldsymbol{y}$. The variables $\boldsymbol{x}$ are the so-called *distinguished variables* (which will be bound with constants in the KB), while $\boldsymbol{y}$ are the *non-distinguished* variables (which are existentially quantified). For example, the following simple query $q(w) \leftarrow PhD(w) \wedge takes(w, z)$ asks for PhDs who are taking something.

Given an interpretation $\mathcal{I}$, the conjunctive query $Q(\boldsymbol{x}) \leftarrow conj(\boldsymbol{x}, \boldsymbol{y})$ is interpreted as the set $Q(\boldsymbol{x})^\mathcal{I}$ of tuples $\boldsymbol{o}$ of elements of $\Delta^\mathcal{I}$ such that, when we assign $\boldsymbol{o}$ to $\boldsymbol{x}$, the first-order formula $\exists \boldsymbol{y}.conj(\boldsymbol{x}, \boldsymbol{y})$ evaluates to true in $\mathcal{I}$.

The reasoning service we are interested in is *(conjunctive) query answering*: given a knowledge base $\mathcal{K}$ and a conjunctive query $Q(\boldsymbol{x})$ over $\mathcal{K}$, compute the *certain answers* to $Q(\boldsymbol{x})$ over $\mathcal{K}$, i.e., the tuples $\boldsymbol{d}$ of constants in $\mathcal{K}$ such that $\boldsymbol{d}^\mathcal{I} \in Q(\boldsymbol{x})^\mathcal{I}$ for every model $\mathcal{I}$ of $\mathcal{K}$. We observe that query answering (properly) generalizes a well known reasoning service in DLs, namely *instance checking*, i.e., logical implication of an ABox assertion. In particular, instance checking can be expressed as the problem of answering (boolean) conjunctive queries constituted by just one ground atom.

2.2 Explanations

It is widely accepted that an explanation corresponds to a formal *proof*. A formal proof is constructed from premises using rules of inference. Although [20] suggests a specific XML-based syntax for inference rule schemas to be used in constructing proofs, we will use the more concise notation used in Programming Languages, and first applied to DLs in [21], which is illustrated in the following inference rule, expressing in one way the transitivity of the $\sqsubseteq$ relationship:

$$\textbf{Isa-trans}\ \frac{\mathcal{T} \vdash B_1 \sqsubseteq B_2 \quad \mathcal{T} \vdash B_2 \sqsubseteq B_3}{\mathcal{T} \vdash B_1 \sqsubseteq B_3}\ B_1, B_2, B_3 \text{ concepts}$$

Here, the name of the rule schema is **Isa-trans**; the antecedent requires that from the TBox $\mathcal{T}$ one can deduce $B_1 \sqsubseteq B_2$ and also $B_2 \sqsubseteq B_3$; the consequent allows one to also deduce from the same TBox that $B_1 \sqsubseteq B_3$; the side-condition of the rule requires B_1, B_2, and B_3 to be concept expressions.

The rules of inference used and the proof itself have certain intuitively desirable properties as far as the understandability of the resulting explanation[5]. These include:

[5] Ideally, empirical user studies would support these claims; as it is, we rely on our and the readers' intuitions.

- *Simplicity*: while some rules of inference (e.g., concluding p from q and "if q then p") are self-evident, others may be so complex that in explaining an inference step one needs to also explain the validity of the inference rule, in addition to explaining the antecedents. Such rules should be avoided, if possible.
- *Brevity*: all things being equal, shorter proofs are preferred, since they take less time to present and understand; note that one way in which to make proofs shorter is to find portions, called lemmas, that are re-used.

Note that the above principles may conflict (e.g., a simpler proof may be longer), and therefore in general there is likely to be no single "ideal" explanation system – more likely one with "knobs" that can be adjusted.

Although not frequently articulated, an explanation involves not only a proof but also a *proof presentation strategy*. For example, there is a decided preference for tree-shaped proofs, produced by rules of inference with a single conclusion, and zero or more antecedents. So-called "natural deduction proofs" produce such proofs, and many explanation facilities for description logics and the semantic web follow these principles (e.g., [10,17,22]). One of the advantages of such proofs is that they support interactive and gradual unfolding of only relevant parts under user control. These ideals are exemplified by Horn logic, where the explanation of goal g provides as a first step the rule $p \wedge q \rightarrow g$ from which g was deduced, and then allows the user to choose follow-up questions concerning the derivation of p and/or q.

While it is possible to present proofs using a very mechanical approach, which produces the same format for all rules of inference, this is not a necessity, and flexibility can lead to improvements. For example, most inference systems from sets of axioms have a reiteration rule of the form

$$\textbf{Given } \frac{}{\mathcal{T}, \psi \vdash \psi} \ \psi \text{ any axiom}$$

which allows any axiom from the theory to be used in a proof. It is better to replace this by a scheme where all axioms in $\mathcal{T}$ are numbered, and whenever some other inference rule uses ψ as an antecedent, ψ is listed, with its number as justification. More interestingly, certain sub-proofs may be judged to be too trivial/obvious, and can therefore be eliminated from the proof when presented as an explanation. A simple example of this involves conjunction exploitation, where we simply allow a proof to use axiom $p \wedge q$, when what is required is p. In a similar vein, in [10] some kinds of inheritance were explained by indicating the ancestor from which the inherited constraint was obtained, without explicitly listing all the intermediate concepts through which inheritance passed.

Of course, there is also the possibility of generating graphical explanation proof trees, or natural language text [23].

3 Explanations of Standard Inferences

3.1 Modified Syntax of *DL-Lite*$_{\mathcal{A}}$

A number of notions in *DL-Lite*$_{\mathcal{A}}$ (and their notation), such as existential constraints, inverses of roles, and complements of concepts or roles are, in our opinion, mathematically

too sophisticated for users familiar only with notations like UML diagrams[6]. For this reason, we propose to alter the surface syntax shown to users.

As far as $\neg$ is concerned, we observe that the restricted occurrence of negated concepts (resp., roles) in *DL-Lite*$_{\mathcal{A}}$, in axioms of the form $B_1 \sqsubseteq \neg B_2$ (resp., $Q_1 \sqsubseteq \neg Q_2$), means that these are really only used to describe that two (un-negated) concepts (resp., roles) are *disjoint*. Hence, we replace each assertion of the form $B_1 \sqsubseteq \neg B_2$ (resp., $Q_1 \sqsubseteq \neg Q_2$) by the assertion $(\mathsf{disjoint}\ B_1\ B_2)$ (resp., $(\mathsf{disjoint}\ Q_1\ Q_2)$), having the same semantics. This eliminates $\neg$ from our axioms, and also from our explanations.

Next, we propose to eliminate the notations $\exists P$ and $\exists P^-$, and replace them by the more familiar notions of "the current domain" and "the current range of role P", respectively, written as $\mathsf{dom}(P)$ and $\mathsf{rng}(P)$[7]. As a result of the above simplifications, concept inclusion assertions 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.

The above transformation also has the desirable effect of eliminating role inverses from concept inclusions. The remaining use of role inverses is in functionality assertions of the form $(\mathsf{funct}\ P^-)$, and in role inclusion assertions. In general, ontology designers are encouraged to declare names for inverse roles (as in UML and OWL 1.1) by using an assertion of the form $(\mathsf{inverseRoles}\ P\ idForInvOfP)$ (e.g., $(\mathsf{inverseRoles}\ makes\ madeBy)$) and then using $idForInvOfP$ instead of P^-.

Unfortunately, this will not allow us to completely ignore the role inverse notation, as illustrated by the following example.

Example 2. Suppose the TBox $\mathcal{T}$ contains the role inclusion assertions:

$$P_1 \sqsubseteq P_2^- \qquad P_2 \sqsubseteq P_3 \qquad P_1 \sqsubseteq P_4^- \qquad P_4 \sqsubseteq P_5 \qquad P_3 \sqsubseteq \neg P_5$$

Observe that P_1 is unsatisfiable in $\mathcal{T}$. The reason is that the first two inclusions imply (by **Isa-trans** and **IsaInv** below) that $P_1 \sqsubseteq P_3^-$; similarly, the next two inclusions imply that $P_1 \sqsubseteq P_5^-$. By the last inclusion, P_3 and P_5 are disjoint, and hence so are P_3^- and P_5^-. Hence, P_1, being subsumed by two disjoint roles, is unsatisfiable. Note that in this proof we referred to P_3^- and P_5^- in explaining the unsatisfiability of P_1, even though neither of these role inverses appears in the TBox. ∎

Therefore, we will in general not be able to avoid the need of confronting the user with role inverses, when she has not specified an alternate name for the inverse of a role[8].

In the following subsections we present the rules of inference required for sound and complete reasoning about a variety of judgments. Because these will be relatively simple, we will not spend any time on issues of proof presentation.

3.2 TBox Reasoning

Subsumption reasoning in *DL-Lite*$_{\mathcal{A}}$ is a particularly simple form of structural subsumption, in part because there are no nested concepts. Therefore, one does not need

[6] Ideally, this would be supported by experimental results.

[7] 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.

[8] We might consider prompting the user for an explicit name for the inverses that are needed before any particular explanation is begun.

any of the complications suggested in [10], such as atomic concepts, normalized concepts, etc.; the standard **IsA**-inference rules for reiteration (givens), reflexivity, and transitivity suffice. The rule for givens is shown in Section 2.2, as is the transitivity rule for concepts; we assume that the latter rule applies also to roles. The reflexivity rule is:

$$\textbf{Isa-refl}\ \frac{}{\mathcal{T} \vdash X \sqsubseteq X}\ X \text{ a concept or a role}$$

We also need inference rules to relate the domains and ranges of a role and its inverse. In such rules (and ones further one), Q denotes a (basic) role, i.e., either an atomic role P or the inverse P^- of an atomic role. Moreover, we assume the syntactic simplification $(P^-)^- = P$.

$$\textbf{Dom-rng-inv}\ \frac{}{\mathcal{T} \vdash \mathsf{dom}(Q) \sqsubseteq \mathsf{rng}(Q^-)}\ Q \text{ a role}$$

$$\textbf{Rng-dom-inv}\ \frac{}{\mathcal{T} \vdash \mathsf{rng}(Q) \sqsubseteq \mathsf{dom}(Q^-)}\ Q \text{ a role}$$

Finally, the following inference rules take into account role subsumption when considering domains, ranges, and inverses:

$$\textbf{Isa-dom}\ \frac{\mathcal{T} \vdash Q_1 \sqsubseteq Q_2}{\mathcal{T} \vdash \mathsf{dom}(Q_1) \sqsubseteq \mathsf{dom}(Q_2)}\ Q_1, Q_2 \text{ roles} \qquad \textbf{Isa-rng}\ \frac{\mathcal{T} \vdash Q_1 \sqsubseteq Q_2}{\mathcal{T} \vdash \mathsf{rng}(Q_1) \sqsubseteq \mathsf{rng}(Q_2)}\ Q_1, Q_2 \text{ roles}$$

$$\textbf{IsaInv}\ \frac{\mathcal{T} \vdash Q_1 \sqsubseteq Q_2}{\mathcal{T} \vdash Q_1^- \sqsubseteq Q_2^-}\ Q_1, Q_2 \text{ roles}$$

Let us now introduce the $\bot$ symbol, denoting the empty set in all interpretations[9]. By the definition of $\sqsubseteq$, we therefore have the following additional inference rule:

$$\textbf{Nothing}\ \frac{}{\mathcal{T} \vdash \bot \sqsubseteq X}\ X \text{ a concept or a role}$$

By definition, an *unsatisfiable/inconsistent* concept or role must be subsumed by $\bot$. In any DL, a concept or role can be shown to be unsatisfiable *indirectly*, by finding, via **Isa-trans**, a superconcept that itself is "directly" unsatisfiable. In *DL-Lite*$_\mathcal{A}$, a concept will be said to be *"directly"* unsatisfiable due to subsumption by disjoint concepts, or due to being the current domain or range of an unsatisfiable role (which fall out of rules **Isa-dom** and **Isa-rng**, when $Q_2 = \bot$), or subsumption by another unsatisfiable concept. Similarly, a role can only be unsatisfiable due to subsumption by another unsatisfiable role, subsumption by disjoint roles, or due to its current domain or range being unsatisfiable. Hence, we need the following inference rules:

$$\textbf{Inc-disj}\ \frac{\mathcal{T} \vdash X \sqsubseteq X_1 \qquad \mathcal{T} \vdash X \sqsubseteq X_2}{\mathcal{T}, (\mathsf{disjoint}\ X_1\ X_2) \vdash X \sqsubseteq \bot}\ X, X_1, X_2 \text{ concepts or roles}$$

$$\textbf{Inc-role-d}\ \frac{\mathcal{T} \vdash \mathsf{dom}(P) \sqsubseteq \bot}{\mathcal{T} \vdash P \sqsubseteq \bot}\ P \text{ atomic role} \qquad \textbf{Inc-role-r}\ \frac{\mathcal{T} \vdash \mathsf{rng}(P) \sqsubseteq \bot}{\mathcal{T} \vdash P \sqsubseteq \bot}\ P \text{ atomic role}$$

In the absence of unsatisfiability, all reasoning about $\sqsubseteq$ reduces to simple classification of atomic concepts and expressions denoting the current domains/ranges of roles,

[9] This will serve as both the empty concept and the empty role, and we assume the convention that $\mathsf{dom}(\bot) = \bot$, $\mathsf{rng}(\bot) = \bot$, and $\bot^- = \bot$.

i.e., computing the so-called Hasse diagram $\mathcal{G}_\mathcal{T}$ induced by the $\sqsubseteq$ assertions in the TBox $\mathcal{T}$. Once this is done, explaining $B_1 \sqsubseteq B_2$ for satisfiable concepts involves only finding the shortest path between them.

The proof of unsatisfiability of a concept or role, though polynomially computable, can in fact be quite cumbersome because it can involve a chain of alternate demonstrations of role unsatisfiability and concept unsatisfiability, connected by the unsatisfiability of role domains and ranges. Moreover, in order to find *shortest proofs*, one needs to consider both indirect and direct ways of showing unsatisfiability. For lack of space, we consider here only the core of the algorithm searching for proofs of direct concept unsatisfiability w.r.t. a TBox $\mathcal{T}$ without unsatisfiable roles.

If a concept B is unsatisfiable w.r.t. $\mathcal{T}$, there must be $\sqsubseteq$-paths from B to two concepts, say X, Y, asserted to be disjoint. For shortest proofs, we require that the sum of the lengths of these paths be minimal. To find this, an algorithm can start from B and explore in *breadth-first order* paths to $\sqsubseteq$-ancestors X and Y until two disjoint such concepts are found (see Figure 1, where parents(X,$\mathcal{T}$) returns the set of concepts Y such that $X \sqsubseteq Y$ is in $\mathcal{T}$, while disjoint(X,Y,$\mathcal{T}$) is a predicate that returns true if $(\mathsf{disjoint}\ X\ Y)$ is in $\mathcal{T}$). Supposing that the lengths of the paths to these concepts are n and m respectively, with $n \geq m$, then the length of this explanation is $n + m$.[10] Unfortunately, this may not be the shortest explanation: if there exist disjoint concepts X' and Y' that are, respectively, j and k steps away from B, these yield an explanation of length $j + k$, which could be less than $n + m$ even if $j > n$, as long as $0 < k < n + m - j$. However, once we have detected the first pair X, Y at distance $n + m$, to detect the shortest explanation, we only have to search up to the limit when $j = n + m$. Adapting the algorithm in Figure 1 for this task is straightforward, as is keeping track of the paths leading from B to X and Y. Interestingly, the algorithm will also find indirect proofs of unsatisfiability, where the paths from B share an initial fragment π; these have a shorter presentation, omitting one of the π.

3.3 ABox Reasoning

In *DL-Lite*$_\mathcal{A}$, one infers new facts about existing individuals by applying inclusion axioms on concepts and roles, and recognizing that $P(a, b)$ entails $\mathsf{dom}(P)(a)$ and $\mathsf{rng}(P)(b)$. This is formalized by the following inference rules:

$$\textbf{Subconcept}\ \frac{\mathcal{T} \vdash B_1 \sqsubseteq B_2 \qquad \langle \mathcal{T}, \mathcal{A} \rangle \vdash B_1(a)}{\langle \mathcal{T}, \mathcal{A} \rangle \vdash B_2(a)}\ \begin{array}{l} B_1, B_2 \text{ concepts;} \\ a \text{ an individual} \end{array}$$

$$\textbf{Subrole}\ \frac{\mathcal{T} \vdash P_1 \sqsubseteq P_2 \qquad \langle \mathcal{T}, \mathcal{A} \rangle \vdash P_1(a, b)}{\langle \mathcal{T}, \mathcal{A} \rangle \vdash P_2(a, b)}\ \begin{array}{l} P_1, P_2 \text{ atomic roles;} \\ a, b \text{ individuals} \end{array}$$

$$\textbf{Subrole-inv}\ \frac{\mathcal{T} \vdash P_1 \sqsubseteq P_2^- \qquad \langle \mathcal{T}, \mathcal{A} \rangle \vdash P_1(a, b)}{\langle \mathcal{T}, \mathcal{A} \rangle \vdash P_2(b, a)}\ \begin{array}{l} P_1, P_2 \text{ atomic roles;} \\ a, b \text{ individuals} \end{array}$$

$$\textbf{Dom-intro}\ \frac{\langle \mathcal{T}, \mathcal{A} \rangle \vdash P(a, b)}{\langle \mathcal{T}, \mathcal{A} \rangle \vdash \mathsf{dom}(P)(a)}\ \begin{array}{l} P \text{ an atomic role;} \\ a, b \text{ individuals} \end{array}$$

[10] For simplicity, in the algorithm we assume that the set of concept inclusion assertions contains no cycle. Such cycles would make all involved concepts equivalent. Either they are detected a priori, or we would need to add to the algorithm a loop-checking condition.

$$\textbf{Rng-intro}\ \frac{\langle \mathcal{T}, \mathcal{A}\rangle \vdash P(a,b)}{\langle \mathcal{T}, \mathcal{A}\rangle \vdash \mathsf{rng}(P)(b)}\quad \begin{array}{l} P \text{ an atomic role;} \\ a, b \text{ individuals} \end{array}$$

To detect unsatisfiability in a KB, one simply looks for objects belonging to concepts which can be deduced to be subsumed by $\bot$, or for objects violating functionality constraints. The one nontrivial aspect is when we prefer shorter explanations. In the case of unsatisfiable 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 unsatisfiability, assuming that the graph also has instance as well as subclass edges.

We note that while the above looks for evidence of knowledge base unsatisfiability, this is not the same problem as diagnosing errors in the knowledge base. Pinpointing [13], and related orthogonal techniques are much more likely to be useful for this task.

3.4 Reasoning in Finite Models

As mentioned, *DL-Lite*$_\mathcal{A}$ does not enjoy the finite model property, and hence inferences that hold specifically in finite models require to be explained.

Example 3. Consider the following TBox

$$\begin{array}{rcl@{\qquad\qquad}rcl} (\mathsf{funct}\ \mathit{tutors}) & & & \mathit{Student} & \sqsubseteq & \mathsf{rng}(\mathit{tutors}) \\ \mathsf{dom}(\mathit{tutors}) & \sqsubseteq & \mathit{TA} & \mathit{TA} & \sqsubseteq & \mathit{Student} \end{array}$$

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

Clearly, the above pattern can be generalized by replacing tutors with the composition of an arbitrary set of roles $Q_1 \circ Q_2 \circ \cdots \circ Q_k$, obtaining rule **Same-cardinality**:

$$\frac{\mathcal{T} \vdash (\mathsf{funct}\ Q_1 \circ \cdots \circ Q_k) \qquad \mathcal{T} \vdash B_2 \sqsubseteq \mathsf{rng}(Q_1 \circ \cdots \circ Q_k) \quad \mathcal{T} \vdash \mathsf{dom}(Q_1 \circ \cdots \circ Q_k) \sqsubseteq B_1 \qquad \mathcal{T} \vdash B_1 \sqsubseteq B_2}{\mathcal{T} \vdash B_2 \sqsubseteq B_1}\quad \begin{array}{l} B_1, B_2 \text{ concepts;} \\ Q_1, \ldots, Q_k \text{ basic roles} \end{array}$$

The remaining question is how one can deduce properties of a composition of roles, given only *DL-Lite*$_\mathcal{A}$ axioms. First, the following rule captures that if all roles are functions, then their composition will be a function:

$$\textbf{Func-comp}\ \frac{}{\mathcal{T}, (\mathsf{funct}\ Q_1), \ldots, (\mathsf{funct}\ Q_k) \vdash (\mathsf{funct}\ Q_1 \circ \cdots \circ Q_k)}\quad \begin{array}{l} Q_1, \ldots, Q_k \\ \text{basic roles} \end{array}$$

And since the current domain of a composition is contained in the current domain of the first role, we also have:

$$\textbf{Dom-comp}\ \frac{\mathcal{T} \vdash \mathsf{dom}(Q_1) \sqsubseteq B_1}{\mathcal{T} \vdash \mathsf{dom}(Q_1 \circ \cdots \circ Q_k) \sqsubseteq B_1}$$

However, $B_2 \sqsubseteq \mathsf{rng}(Q_1 \circ \cdots \circ Q_k)$ does not follow from $B_2 \sqsubseteq \mathsf{rng}(Q_k)$ alone, because the *current* range of the composition may be smaller, if not all values in $\mathsf{dom}(Q_k)$ are reached by $Q_1 \circ \cdots \circ Q_{k-1}$. So one also needs the entire current domain of Q_k to be contained in the current range of $Q_1 \circ \cdots \circ Q_{k-1}$, leading to the rule:

$$\textbf{Rng-comp}\ \frac{\mathcal{T} \vdash \mathsf{dom}(Q_k) \sqsubseteq \mathsf{rng}(Q_1 \circ \cdots \circ Q_{k-1}) \quad \mathcal{T} \vdash B_2 \sqsubseteq \mathsf{rng}(Q_k)}{\mathcal{T} \vdash B_2 \sqsubseteq \mathsf{rng}(Q_1 \circ \cdots \circ Q_k)}\ k \geq 2$$

It follows from results in [24] that these are *all* the possible additional subsumption inferences needed for the finite model case.

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 Answers to Conjunctive Queries over a *DL-Lite* ABox

Consider first the simpler 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, this means exhibiting the values used for the existentially quantified variables in the body of the query. For example, if `MIMI` is an answer to query

$$Q_0(x) \leftarrow \mathit{Student}(x), \mathit{supervisedBy}(x, y), \mathit{teaches}(y, z)$$

one would need to locate some "witness" values `ANNA` and `ENG101` for variables y and z, and then explain that $\mathit{Student}(\texttt{MIMI}), \mathit{supervisedBy}(\texttt{MIMI}, \texttt{ANNA})$ and $\mathit{teaches}(\texttt{ANNA}, \texttt{ENG101})$ are atoms present in the database. In general, it is possible that a value/tuple[11] appears in the answer for multiple reasons. In the above example, there may be alternate bindings of y and z, which together with $x = \texttt{MIMI}$, satisfy the query. In these case the user needs to be given the option of seeing an enumeration of the different explanations. The principle of minimality would not seem to enter into the choice of explanations here because all explanations are identical in form.

In the case of *DL-Lite*$_\mathcal{A}$, 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 $\mathit{Student}(\texttt{MIMI})$ and $\mathit{Student} \sqsubseteq \mathit{Person}$, then we must also add $\mathit{Person}(\texttt{MIMI})$; and if we have $\mathit{Professor}(\texttt{GINA})$ and $\mathit{Professor} \sqsubseteq \mathsf{dom}(\mathit{teaches})$, then one can conclude

[11] In this example, and the rest of this paper, we deal only with queries that return a single value. This is only a presentation strategy – the theory and implementation we present applies equally to queries that have multiple variables in the head.

that there is some (hypothetical) individual, say $@c$, representing what GINA teaches, and that $teaches(\texttt{GINA}, @c)$ holds. Such hypothetical individuals may also get additional properties of their own. Unfortunately, the result can be an infinite database since the axioms may contain cyclic dependencies; e.g., $Person \sqsubseteq \mathsf{dom}(hasParents)$, $\mathsf{rng}(hasParents) \sqsubseteq Person$.

From theoretical results [6], we know that from the TBox $\mathcal{T}$ and the ABox $\mathcal{A}$ one can derive a (generally infinite) canonical model $can(\mathcal{T}, \mathcal{A})$[12], by introducing "hypothetical individuals", whose existence is posited by axioms, as illustrated in the example above. A crucial property of $can(\mathcal{T}, \mathcal{A})$ is that each conjunctive query $Q(x)$ 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 $Q(x)$. 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 justify 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 the algorithm for query answering in *DL-Lite*$_\mathcal{A}$. Query answering in *DL-Lite*$_\mathcal{A}$ [6] is performed by first rewriting the original query $Q(x)$ into a set $\mathcal{S} = \{Q_0(x), \ldots, Q_n(x)\}$ of alternate queries, then evaluating these over the original ABox (treated as a closed database), and finally returning the union of the results. Each query in $\mathcal{S}$ expresses *necessary* conditions on values x to satisfy the original query $Q(x)$, and the entire set $\mathcal{S}$ has the property that the answer to the original query $Q(x)$ is the union of the answers to the queries in $\mathcal{S}$ when executed over the ABox.

Example 4. Consider the following TBox $\mathcal{T}$

$$PhD \sqsubseteq Student \quad (5) \qquad \mathsf{rng}(supervisedBy) \sqsubseteq Professor \quad (7)$$
$$PhD \sqsubseteq \mathsf{dom}(supervisedBy) \quad (6) \qquad Professor \sqsubseteq \mathsf{dom}(teaches) \quad (8)$$

and the query $Q_0(x) \leftarrow Student(x), supervisedBy(x, y), teaches(y, z)$. The *DL-Lite*$_\mathcal{A}$ rewriting algorithm would rewrite Q_0 into the following set of queries:

$$\begin{aligned}
&Q_0(x) \leftarrow Student(x), supervisedBy(x, y), teaches(y, z)\\
&Q_1(x) \leftarrow PhD(x), supervisedBy(x, y), teaches(y, z)\\
&Q_2(x) \leftarrow PhD(x), supervisedBy(x, y), Professor(y)\\
&Q_3(x) \leftarrow PhD(x), supervisedBy(x, y), supervisedBy(w, y)\\
&Q_4(x) \leftarrow PhD(x), supervisedBy(x, y)\\
&Q_5(x) \leftarrow PhD(x), PhD(x)\\
&Q_6(x) \leftarrow PhD(x)
\end{aligned}$$

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 (5). Similarly, $Q_2(x)$ is obtained from $Q_1(x)$ by making use of axiom (8), which can be seen

[12] The canonical model corresponds to what in databases is called *chase* of a database w.r.t. a set of constraints [6].

in FOL as $\forall v.\exists w.Professor(v) \leftarrow teaches(v,w)$ [13]. On the other hand, a unification step collapses two atoms with the same predicate when the corresponding arguments can be unified; e.g., query $Q_4(x)$ is obtained from $Q_3(x)$ by unifying the two atoms $supervisedBy(x,y)$ and $supervisedBy(w,y)$. Unifications are essential in order to enable replacements where a variable is required to occur only once, as per the previous technical footnote.

Note that the algorithm need not produce a linearly ordered set of rewritings: if the TBox had an additional axiom $MSc \sqsubseteq Student$, there would also be four rewritings $Q_{1b}, \ldots, Q_{4b}$ paralleling $Q_1, \ldots, Q_4$, with MSc replacing PhD. ∎

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 certain individual is in the answer to the original query. Suppose that an individual b is part of the answer to $Q(x)$, and we want to explain why. 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. Since $Q(b)$ was true, we select from $\mathcal{S}$ the rewritings that produce b as an answer when directly evaluated over the ABox (viewed as a closed database)[14]. Let $Q_k(x)$ be one such rewriting. We (re)compute the derivation of $Q_k(x)$ from $Q(x)$, building a data structure that tracks how (changed) atoms in one query are derived from the predecessor query. This is easy for substitutions, if we replace each atom in the query by a stack/list whose top is the most recently rewritten form of the original atom. For unifications, we put a pointer on the stack from one of the unified atoms to the stack of the other atom.

In our example, BOB would be returned by both $Q_5(x)$ and $Q_6(x)$, so the derivation of $Q_5(x)$ and $Q_6(x)$ from $Q(x)$ is reconstructed. The derivation of Q_5 would yield roughly the following data structure:

```
[ [ PhD(x), axiom1, Student(x) ],
  [ PhD(x), axiom2, supervisedBy(x,y) ],
  [ pointer(atom(2)), unify(w,x), supervisedBy(w,y),
    axiom3, Professor(y), axiom4, teaches(y,z) ] ]
```

Step 2. Since $Q_k(b)$ is true, there is an assignment θ of ABox individuals to the variables in the head and body of $Q_k(x)$ such that $\theta(x) = b$, and for each atom β in $Q_k(x)$, $\theta(\beta)$ is an ABox fact.

In our example, the ABox contains $PhD(\texttt{BOB})$, and for $Q_5(x)$ (or $Q_6(x)$), we would start with $\theta(x) = \texttt{BOB}$.

Step 3. Traversing backwards the sequence of rewritings from $Q_k(x)$ to $Q(x)$, we extend the substitution θ to the variables in the intervening queries, by keeping track of unifications, or assigning to such variables newly introduced *Skolem constants* (corresponding to objects introduced by the inclusion assertions). More precisely, when going

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

[14] By completeness of the query answering algorithm [6], at least one such rewriting $Q_k(x)$ will always exist.

$supervisedBy(\texttt{BOB}, \texttt{@1})$ (*)
{ by meaning of domain, from
$\mathsf{dom}(supervisedBy)(\texttt{BOB})$
{ by **Subconcept** rule from
$PhD \sqsubseteq \mathsf{dom}(supervisedBy)$ { Axiom 2 }
$PhD(\texttt{BOB})$ { DB fact } } }

Fig. 3. Domain rule expansion

backwards from Q_i to Q_{i-1}, from which Q_i was generated, if no variable has been eliminated when rewriting Q_{i-1} to Q_i, then θ need not be extended. When a variable z has been eliminated because it has been unified with a variable y, then we set $\theta(z) = \theta(y)$. While when a variable z has been eliminated in Q_i by replacing an atom $R(y, z)$ (resp., $R(z, y)$) with $A(y)$, due to inclusion axiom $A \sqsubseteq \mathsf{dom}(R)$ (resp., $A \sqsubseteq \mathsf{rng}(R)$), then we set $\theta(z) = @c$, where $@c$ is a fresh constant representing a new hypothetical individual. In this way, when one reaches the original query $Q(x)$, θ 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 $\theta(Q(x))$, i.e., the set of atoms of the original query to which θ has been applied.

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

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

Step 4. The data structure constructed in Step 1, together with the substitutions gathered in Step 1 unifications, as well as Steps 2 and 3, yields a complete proof tree of the atoms in the original query (see Figure 2). Note that the leftmost part of this proof tree (its first level) corresponds to the initial explanation we suggested in Step 3. In an interactive session, users can control the iterative expansion of this proof tree to lower depths in whatever order they desire. It is interesting to note that pointers in the data structure become Lemmas – previously justified atoms, thereby providing a much shorter global proof. So unification in the rewriting algorithm has an interesting benefit for explanation.

We observe that an explanation always exists, and that the above algorithm will *always* provide one. Note also that one can manipulate here the proof tree, to make proof steps more understandable to humans. For example, in this case one can expand the rules dealing with domain and range. This could lead to the modified proof fragment depicted in Figure 3.

Several issues need to be addressed at this point. First is the selection of the rewriting(s) $Q_k(x)$ from $\mathcal{S}$ in Step 1. For this, we need an efficient way of finding only the indexes j of those rewritings that actually make $Q(b)$ be true. 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. Originally, if $\mathcal{S} = \{Q_1(x), \ldots, Q_n(x)\}$, then the rewriting would normally have been written in SQL as:

```
SELECT x FROM ... WHERE Q1(x)  UNION
SELECT x FROM ... WHERE Q2(x)  UNION ...
```

We would instead use the query

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

Secondly, in order to execute Step 3, for each of the tag values j actually occurring in the answer, we need to get the ABox tuples that would contribute to making the body of query $Q_j(b)$ be true. In our example, if the ABox was $\{PhD(\texttt{BOB}), Student(MIMI), supervisedBy(\texttt{BOB}, \texttt{ALICE})\}$, then tags 4, 5 and 6 would all be returned for answer $x = BOB$ of $Q(x)$, and we would obtain the set of atoms $\{PhD(\texttt{BOB}), supervisedBy(\texttt{BOB}, \texttt{ALICE})\} \cup \{PhD(\texttt{BOB})\} \cup \{PhD(\texttt{BOB})\}$.

Note that in case more than one rewriting returns the answer b, and the rewritings are not reducible one to the other by unification, then we have alternate explanations for why $Q(b)$ holds. In this case, it seems that explanations involving fewer rewritings are preferable since they involve less abstract reasoning. In our example, the explanation based on Q_4, shown in Figure 4, seems clearly preferable to the previous explanation for $Q_5(BOB)$ because it is shorter (involving fewer rewritings of the original query). All things being equal, we also believe an explanation would be preferred if it introduces fewer hypothetical (Skolem) individuals.

Moreover, Q_5 is preferable to Q_6 because both atoms would be supported by grounding to the same atomic value, thereby saving a unification step in the explanation, which would have resulted in a pointer/Lemma call in the explanation.

Therefore, to find the better rewritings first, we need a search strategy that tests whether a rewriting is satisfied in the database of atoms selected above, before applying any other replacement or unification. And one which uses replacements that introduce Skolem constants as late as possible.

We have implemented the above-described algorithm for generating explanations in a prototype Prolog program, which exploits Prolog's unification technique (for unifying conjunctive query atoms, testing applicability of axiom replacements, and for finding database substitutions), and its backtracking control structure (through an invertible

Student(BOB)
{ by **Subconcept** rule from
PhD $\sqsubseteq$ *Student* { Axiom 1 }
PhD(BOB) { DB fact } }
supervisedBy(BOB, ALICE)
teaches(ALICE, @1)
{ by **Subconcept** rule from
Professor $\sqsubseteq$ dom(*teaches*) { Axiom 4 }
Professor(@1)
{ by **Subconcept** rule from
rng(*supervisedBy*) $\sqsubseteq$ *Professor* { Axiom 3 }
supervisedBy(BOB, ALICE) { DB fact } } }

Fig. 4. Alternate (shorter) explanation for $Q(\texttt{BOB})$

append predicate) to search through all possible sequences of rewritings of length less than $n =$ max{tag value returned by the query}.

5 Conclusions

The paper tackles the problem of explaining *DL-Lite* reasoning, viewed in part as (*i*) requiring inference rules for building proofs, and (*ii*) finding short proofs. (The use of inference rules makes possible, among others, a connection to generic explanation software available on the Semantic Web [17].) In general, an alternate, more accessible syntax is introduced for *DL-Lite*, and an algorithm for finding shortest proofs of inconsistency is presented. Of greater novelty and complexity is the *explanation* of reasoning in finite models, which is particularly relevant for conceptual models of databases.

Since *DL-Lite* is intended to support efficient conjunctive query evaluation over a DL KB, we address for the first time explanation for this. In particular, we provide a theoretically sound technique and a prototype implementation for finding explanations of the fact that some value b is returned as an answer to query $Q(x)$ over knowledge base $(\mathcal{T}, \mathcal{A})$. These explanations are minimal length in the sense that fewest transformations steps are applied from the original query. The performance system used for query-answering is used to retrieve a minimal set of tuples needed to explain the truth of $Q(b)$, thus making the explanation component scalable even for very large ABoxes.

Future work includes a component for finding *all* different explanations for some conclusion (which is useful for the case when the conclusion is no longer desired), and especially explaining why a value e was *NOT* returned by a conjunctive query.

Acknowledgments. This research has been partially supported by the EU FP6 FET project TONES (Thinking ONtologiES) under contract number FP6-7603, by the Italian PRIN 2006 project NGS (New Generation Search), funded by MIUR, and by the U.S. DHS under ONR grant N00014-7-1-0150.

References

1. Borgida, A., Calvanese, D., Rodriguez-Muro, M.: Explanation in DL-Lite. In: Proc. of the 2008 Description Logic Workshop (DL 2008). CEUR Electronic Workshop Proceedings (2008), http://ceur-ws.org/
2. Elmasri, R., Wiederhold, G.: GORDAS: A formal high-level query language for the entity-relationship model. In: Proc. of the 2nd Int.Conf.on the Entity-Relationship Approach (ER 1981), pp. 49–72 (1981)
3. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge (2003)
4. Calvanese, D., Lenzerini, M., Nardi, D.: Unifying class-based representation formalisms. J. of Artificial Intelligence Research 11, 199–240 (1999)
5. Berardi, D., Calvanese, D., De Giacomo, G.: Reasoning on UML class diagrams. Artificial Intelligence 168(1–2), 70–118 (2005)
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(3), 385–429 (2007)

7. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tailoring OWL for data intensive ontologies. In: Proc. of the Workshop on OWL: Experiences and Directions (OWLED 2005). CEUR Electronic Workshop Proceedings, vol. 188 (2005), http://ceur-ws.org/Vol-188/
8. Shmueli, O., Tsur, S.: Logical diagnosis of LDL programs. In: Proc. of the 7th Int. Conf. on Logic Programming (ICLP 1990), pp. 112–129 (1990)
9. Arora, T., Ramakrishnan, R., Roth, W.G., Seshadri, P., Srivastava, D.: Explaining program execution in deductive systems. In: Ceri, S., Tsur, S., Tanaka, K. (eds.) DOOD 1993. LNCS, vol. 760, pp. 101–119. Springer, Heidelberg (1993)
10. McGuinness, D.L., Borgida, A.: Explaining subsumption in description logics. In: Proc. of the 14th Int. Joint Conf. on Artificial Intelligence (IJCAI 1995), pp. 816–821 (1995)
11. 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, pp. 189–204 (2005)
12. Schlobach, S.: Explaining subsumption by optimal interpolation. In: Alferes, J.J., Leite, J.A. (eds.) JELIA 2004. LNCS (LNAI), vol. 3229, pp. 413–425. Springer, Heidelberg (2004)
13. Schlobach, S.: Debugging and semantic clarification by pinpointing. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 226–240. Springer, Heidelberg (2005)
14. Schlobach, S., Cornet, R.: Explanation of terminological reasoning: A preliminary report. In: Proc.of the 2003 Description Logic Workshop (DL 2003) (2003)
15. Godfrey, P., Minker, J., Novik, L.: An architecture for a cooperative database system. In: Risch, T., Litwin, W. (eds.) ADB 1994. LNCS, vol. 819, pp. 3–24. Springer, Heidelberg (1994)
16. Godfrey, P.: Minimization in cooperative response to failing database queries. Int. J. of Cooperative Information Systems 6, 95–149 (1997)
17. McGuinness, D.L., Pinheiro da Silva, P.: Explaining answers from the Semantic Web: the Inference Web approach. J. of Web Semantics 1(4), 397–413 (2004)
18. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In: Proc.of the 10th Int.Conf.on the Principles of Knowledge Representation and Reasoning (KR 2006), pp. 260–270 (2006)
19. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. on Data Semantics X, 133–173 (2008)
20. Pinheiro da Silva, P., McGuinness, D.L., Fikes, R.: A proof markup language for semantic web services. Information Systems 31(4–5), 381–395 (2006)
21. Borgida, A.: From type systems to knowledge representation: Natural semantics specifications for description logics. J. of Intelligent and Cooperative Information Systems 1(1), 93–126 (1992)
22. Borgida, A., Franconi, E., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F.: Explaining $\mathcal{ALC}$ subsumption. In: Proc.of the 1999 Description Logic Workshop (DL 1999). CEUR Electronic Workshop Proceedings, vol. 22 (1999), http://ceur-ws.org/Vol-22/
23. Fiedler, A.: Natural Language Proof Explanation. In: Hutter, D., Stephan, W. (eds.) Mechanizing Mathematical Reasoning. LNCS, vol. 2605, pp. 342–363. Springer, Heidelberg (2005)
24. Rosati, R.: Finite model reasoning in DL-Lite. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021. Springer, Heidelberg (2008)