DL-Lite: Tractable Description Logics for Ontologies

Diego Calvanese[1], **Giuseppe De Giacomo**[2], **Domenico Lembo**[2], **Maurizio Lenzerini**[2], **Riccardo Rosati**[2]

[1] Faculty of Computer Science
Free University of Bolzano/Bozen
Piazza Domenicani 3
I-39100 Bolzano, Italy
calvanese@inf.unibz.it

[2] Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113
I-00198 Roma, Italy
lastname@dis.uniroma1.it

Abstract

We propose a new Description Logic, called *DL-Lite*, specifically tailored to capture basic ontology languages, while keeping low complexity of reasoning. Reasoning here means not only computing subsumption between concepts, and checking satisfiability of the whole knowledge base, but also answering complex queries (in particular, conjunctive queries) over the set of instances maintained in secondary storage. We show that in *DL-Lite* the usual DL reasoning tasks are polynomial in the size of the TBox, and query answering is polynomial in the size of the ABox (i.e., in data complexity). To the best of our knowledge, this is the first result of polynomial data complexity for query answering over DL knowledge bases. A notable feature of our logic is to allow for a separation between TBox and ABox reasoning during query evaluation: the part of the process requiring TBox reasoning is independent of the ABox, and the part of the process requiring access to the ABox can be carried out by an SQL engine, thus taking advantage of the query optimization strategies provided by current DBMSs.

Introduction

One of the most important lines of research in Description Logics (DLs) is concerned with the trade-off between expressive power and computational complexity of sound and complete reasoning. Research carried out in the past on this topic has shown that many DLs with efficient, i.e., worst-case polynomial time, reasoning algorithms lack modeling power required in capturing conceptual models and basic ontology languages, while most DLs with sufficient modeling power suffer from inherently worst-case exponential time behavior of reasoning [4, 5].

Although the requirement of polynomially tractable reasoning might be less stringent when dealing with relatively small ontologies, we believe that the need of efficient reasoning algorithms is of paramount importance when the ontology system is to manage large amount of objects (e.g., from thousands to millions of instances). This is the case of several important applications where the use of ontologies is advocated nowadays. For example, in the Semantic Web, ontologies are often used to describe the relevant concepts of Web repositories, and such repositories may incorporate very large data sets, which constitute the instances of the concepts in the ontology. In such cases, two requirements emerge that are typically overlooked in DLs. First, the number of objects in the knowledge bases requires managing instances of concepts (i.e., ABoxes) in secondary storage. Second, significant queries to be posed to the knowledge bases are more complex than the simple queries (i.e., concepts and roles) usually considered in DL research. Unfortunately, in these contexts, whenever the complexity of reasoning is exponential in the size of the instances (as for example in Fact[1], Racer[2] and in [11]), there is little hope for effective instance management and query answering algorithms.

In this paper we propose a new DL, called *DL-Lite*, specifically tailored to capture basic ontology languages, while keeping low complexity of reasoning, in particular, polynomial in the size of the instances in the knowledge base. Reasoning here means not only computing subsumption between concepts, and checking satisfiability of the whole knowledge base, but also answering complex queries over the set of instances maintained in secondary storage.

Our contributions are the following:

1. We define *DL-Lite*, and show that it is rich enough to capture a significant ontology language. Although at a first sight *DL-Lite* appears to be a very simple DL, the kind of modeling constructs in our logic makes it suitable for expressing a variety of representation languages widely adopted in different contexts, such as basic ontology languages, conceptual data models (e.g., Entity-Relationship [2]), and object-oriented formalisms (e.g., basic UML class diagrams[3]).
2. For such a DL we propose novel reasoning techniques for a variety of tasks, including conjunctive query answering and containment between conjunctive queries over concepts and roles. Our presentation is focused especially on the problem of answering conjunctive queries over a knowledge base. We observe that this is one of the few results on answering complex queries (i.e., not corresponding simply to a concept or a role) over a DL knowledge base [11]. Indeed, answering conjunctive queries over a knowledge base is a challenging problem, even in the case

[1] http://www.cs.man.ac.uk/~horrocks/FaCT/
[2] http://www.sts.tu-harburg.de/~r.f.moeller/racer/
[3] http://www.omg.org/uml/

of *DL-Lite*, where the combination of constructs expressible in the knowledge base does not pose particular difficulties in computing subsumption. Notice that, in spite of the simplicity of *DL-Lite* TBoxes, the ability of taking TBox knowledge into account during the process of answering conjunctive queries goes beyond the "variable-free" fragments of first-order logic represented by DLs.

3. An important feature of our approach is that it is perfectly suited to representing ABox assertions managed in secondary storage by a Data Base Management System (DBMS). Indeed, our query answering algorithm is based on the idea of expanding the original query into a set of queries that can be directly evaluated by an SQL engine over the ABox, thus taking advantage of well established query optimization strategies. Notably, this was one of the motivations behind several research works done on CLASSIC in the 80's [6].
4. We analyze the complexity of reasoning in *DL-Lite*. We show that the usual reasoning tasks considered in DLs (i.e., subsumption and satisfiability) can be done in polynomial time. As for query answering, computing the answers to a conjunctive query is worst-case exponential in the size of the TBox and the query, but is polynomial in the size of the ABox, i.e., in data complexity [17]. Hence, the complexity of answering queries is no worse than traditional query evaluation in relational databases[4].

A prototype implementation of *DL-Lite* has been developed and tested within a research project carried out jointly by our institution and the IBM Tivoli Laboratory. First experiments show that our approach is extremely effective: complex domains can be modeled in *DL-Lite*, and it takes no more than a few minutes to answer conjunctive queries over knowledge bases with millions of instances.

DL-Lite

As usual in DLs, *DL-Lite* allows for representing the domain of interest in terms of concepts, denoting sets of objects, and roles, denoting binary relations between objects. *DL-Lite* concepts are defined as follows:

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

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

A *DL-Lite* knowledge base (KB) is constituted by two components: a TBox used to represent intensional knowledge, and an ABox, used to represent extensional information. *DL-Lite TBox* assertions are of the form

$B \sqsubseteq C$ *inclusion assertions*
$(\mathsf{funct}\ R), (\mathsf{funct}\ R^-)$ *functionality assertions*

An inclusion assertion expresses that a basic concept is subsumed by a general concept, while a functionality assertion expresses the (global) functionality of a role, or of the inverse of a role.

As for the ABox, *DL-Lite* allows for assertions of the form:

$$B(a), R(a,b) \quad \textit{membership assertions}$$

where a and b are constants. These assertions state respectively that the object denoted by a is an instance of the basic concept B, and that the pair of objects denoted by (a, b) is an instance of the role R.

Although *DL-Lite* is quite simple from the language point of view, it allows for querying the extensional knowledge of a KB in a much more powerful way than usual DLs, in which only membership to a concept or to a role can be asked. Specifically, *DL-Lite* allows for using conjunctive queries of arbitrary complexity. A conjunctive query (CQ) q over a knowledge base $\mathcal{K}$ is an expression of the form

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

where $\vec{x}$ are the so-called *distinguished variables*, $\vec{y}$ are existentially quantified variables called the *non-distinguished* variables, and $conj(\vec{x}, \vec{y})$ is a conjunction of atoms of the form $B(z)$, or $R(z_1, z_2)$, where B and R are respectively a basic concept and a role in $\mathcal{K}$, and z, z_1, z_2 are constants in $\mathcal{K}$ or variables in $\vec{x}$ or $\vec{y}$. Sometimes, for simplifying notation, we will use the Datalog syntax, and write queries of the above form as $q(\vec{x}) \leftarrow body(\vec{x}, \vec{y})$, where the existential quantification $\exists \vec{y}$ has been made implicit, and the symbol "," is used for conjunction in $body(\vec{x}, \vec{y})$.

The semantics of *DL-Lite* is given in terms of interpretations over a fixed infinite *domain* Δ. We assume to have one constant for each object, denoting exactly that object. In other words, we have *standard names* [15], and we will not distinguish between the alphabet of constants and Δ.

An *interpretation* $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ consists of a first order structure over Δ with an *interpretation function* $\cdot^{\mathcal{I}}$ such that:

$$\begin{array}{ll} A^{\mathcal{I}} \subseteq \Delta & R^{\mathcal{I}} \subseteq \Delta \times \Delta \\ (\neg B)^{\mathcal{I}} = \Delta \setminus B^{\mathcal{I}} & (\exists R)^{\mathcal{I}} = \{c \mid \exists c' . (c, c') \in R^{\mathcal{I}}\} \\ (C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} & (\exists R^-)^{\mathcal{I}} = \{c \mid \exists c' . (c', c) \in R^{\mathcal{I}}\} \end{array}$$

An interpretation $\mathcal{I}$ is a *model* of an inclusion assertion $B \sqsubseteq C$ iff $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$; $\mathcal{I}$ is a model of a functionality assertion $(\mathsf{funct}\ R)$ if $(c, c') \in R^{\mathcal{I}} \wedge (c, c'') \in R^{\mathcal{I}} \supset c' = c''$, similarly for $(\mathsf{funct}\ R^-)$; $\mathcal{I}$ is a model of a membership assertion $B(a)$ (resp. $R(a, b)$) if $a \in B^{\mathcal{I}}$ (resp. $(a, b) \in R^{\mathcal{I}}$). A *model of a KB* $\mathcal{K}$ is an interpretation $\mathcal{I}$ that is a model of all the assertions in $\mathcal{K}$. A KB is *satisfiable* if it has at least one model. A KB $\mathcal{K}$ *logically implies* an assertion α if all the models of $\mathcal{K}$ are also models of α. A query $q(\vec{x}) \leftarrow \exists \vec{y}.conj(\vec{x}, \vec{y})$ is interpreted in an interpretation $\mathcal{I}$ as the set $q^{\mathcal{I}}$ of tuples $\vec{c} \in \Delta \times \cdots \times \Delta$ such that when we substitute the variables $\vec{x}$ with the constants $\vec{c}$, the formula $\exists \vec{y}.conj(\vec{x}, \vec{y})$ evaluates to true in $\mathcal{I}$.

Since *DL-Lite* deals with conjunctive queries, the basic reasoning services that are of interest are:

[4] We remind the reader that the algorithms for answering a conjunctive query posed to a relational database are exponential in the size of the query.

- *query answering*: given a query q with distinguished variables $\vec{x}$ and a KB $\mathcal{K}$, return the set $ans(q, \mathcal{K})$ of tuples $\vec{c}$ of constants of $\mathcal{K}$ such that in every model $\mathcal{I}$ of $\mathcal{K}$ we have $\vec{c} \in q^{\mathcal{I}}$. Note that this task generalizes *instance checking* in DLs, i.e., checking whether a given object is an instance of a specified concept in every model of the knowledge base.
- *query containment*: given two queries q_1 and q_2 and a KB $\mathcal{K}$, verify whether in every model $\mathcal{I}$ of $\mathcal{K}$ $q_1^{\mathcal{I}} \subseteq q_2^{\mathcal{I}}$. Note that this task generalizes *logical implication* of inclusion assertions in DLs.
- *KB satisfiability*: verify whether a KB is satisfiable.

Example 1 Consider the atomic concepts $Professor$ and $Student$, the roles $TeachesTo$ and $HasTutor$, and the following *DL-Lite* TBox $\mathcal{T}$:

$$\begin{array}{ll} Professor \sqsubseteq \exists TeachesTo & Student \sqsubseteq \exists HasTutor \\ \exists TeachesTo^- \sqsubseteq Student & \exists HasTutor^- \sqsubseteq Professor \\ Professor \sqsubseteq \neg Student & (\mathsf{funct}\ HasTutor). \end{array}$$

Assume that the ABox $\mathcal{A}$ contains only the assertion $HasTutor(\mathsf{John}, \mathsf{Mary})$. Finally, consider the query $q(x) \leftarrow TeachesTo(x, y), HasTutor(y, z)$, asking for professors that teach to students that have a tutor. ∎

Although equipped with advanced reasoning services, at first sight *DL-Lite* might seem rather weak in modeling intensional knowledge, and hence of limited use in practice. In fact, this is not the case. Despite the simplicity of its language and the specific form of inclusion assertions allowed, *DL-Lite* is able to capture the main notions (though not all, obviously) of both ontologies, and of conceptual modeling formalisms used in databases and software engineering (i.e., ER and UML class diagrams). In particular, *DL-Lite* assertions allow us to specify *ISA*, e.g., stating that concept A_1 is subsumed by concept A_2, using $A_1 \sqsubseteq A_2$; *disjointness*, e.g., between concepts A_1 and A_2, using $A_1 \sqsubseteq \neg A_2$; *role-typing*, e.g., stating that the first (resp., second) component of the relation R is an instance of A_1 (resp., A_2), using $\exists R \sqsubseteq A_1$ (resp., $\exists R^- \sqsubseteq A_2$); *participation constraints*, e.g., stating that all instances of concept A participate to the relation R as the first (resp., second) component, using $A \sqsubseteq \exists R$ (resp., $A \sqsubseteq \exists R^-$); *non-participation constraints*, using $A \sqsubseteq \neg\exists R$ and $A \sqsubseteq \neg\exists R^-$; *functionality restrictions* on relations, using $(\mathsf{funct}\ R)$ and $(\mathsf{funct}\ R^-)$. Notice that DL-Lite is a strict subset of OWL Lite, the less expressive sublanguage of OWL[5], which presents some constructs (e.g., some kinds of role restrictions) that are non expressible in DL-Lite, and that make reasoning in OWL Lite non-tractable in general.

Reasoning in *DL-Lite*

It can be shown that *query containment* can be reformulated as *query answering* using techniques similar to the ones in [1]. Hence, we concentrate on query answering only.

We first address some preliminary issues, and then we define the query reformulation algorithm PerfectRef, which is at the heart of our query evaluation algorithm Answer. Finally, we address correctness and complexity issues.

[5]http://www.w3.org/TR/owl-features

KB normalization We denote by Normalize($\mathcal{K}$) the *DL-Lite* KB obtained by transforming the KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ as follows. The ABox $\mathcal{A}$ is expanded by adding to $\mathcal{A}$ the assertions $\exists R(a)$ and $\exists R^-(b)$ for each $R(a, b) \in \mathcal{A}$.

Then, assertions of $\mathcal{K}$ in which conjunctive concepts occur are rewritten by iterative application of the rule: if $B \sqsubseteq C_1 \sqcap C_2$ occurs in $\mathcal{T}$, then replace it with the two assertions $B \sqsubseteq C_1$, $B \sqsubseteq C_2$.

The TBox $\mathcal{T}$ resulting from such a transformation contains assertions of the form (i) $B_1 \sqsubseteq B_2$, where B_1 and B_2 are basic concepts (i.e., each of them is either an atomic or an existential concept), which we call *positive inclusions (PIs)*; (ii) $B_1 \sqsubseteq \neg B_2$, where B_1 and B_2 are basic concepts, which we call *negative inclusions (NIs)*; (iii) functionality assertions on roles of the form $(\mathsf{funct}\ R)$ or $(\mathsf{funct}\ R^-)$.

Then, the TBox $\mathcal{T}$ is expanded by computing all (non-trivial) NIs between basic concepts implied by $\mathcal{T}$. More precisely, the TBox $\mathcal{T}$ is closed with respect to the following inference rule: if $B_1 \sqsubseteq B_2$ occurs in $\mathcal{T}$ and either $B_2 \sqsubseteq \neg B_3$ or $B_3 \sqsubseteq \neg B_2$ occurs in $\mathcal{T}$ (where B_1, B_2, B_3 are arbitrary basic concepts), then add $B_1 \sqsubseteq \neg B_3$ to $\mathcal{T}$. It can be shown that, after the above closure of $\mathcal{T}$, for every pair of basic concepts B_1, B_2, we have that $\mathcal{T} \models B_1 \sqsubseteq \neg B_2$ iff either $B_1 \sqsubseteq \neg B_2 \in \mathcal{T}$ or $B_2 \sqsubseteq \neg B_1 \in \mathcal{T}$.

It is immediate to verify that, for every *DL-Lite* KB $\mathcal{K}$, Normalize($\mathcal{K}$) is equivalent to $\mathcal{K}$, in the sense that the set of models of $\mathcal{K}$ coincides with that of Normalize($\mathcal{K}$). In the following, without loss of generality we assume that every concept name or role name occurring in $\mathcal{A}$ also occurs in $\mathcal{T}$.

ABox storage Once the ABox is normalized, we store it under the control of a DBMS, in order to effectively manage objects in the knowledge base by means of an SQL engine. To this aim, we construct a relational database which faithfully represents a normalized ABox $\mathcal{A}$. More precisely,

- for each basic concept B occurring in $\mathcal{A}$, we define a relational table tab_B of arity 1, such that $\langle a \rangle \in tab_B$ iff $B(a) \in \mathcal{A}$;
- for each role R occurring in $\mathcal{A}$, we define a relational table tab_R of arity 2, such that $\langle a, b \rangle \in tab_R$ iff $R(a, b) \in \mathcal{A}$.

We denote with DB($\mathcal{A}$) the relational database thus constructed.

KB satisfiability The algorithm Consistent takes as input a normalized KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and verifies the following conditions:
(i) there exists a NI $B_1 \sqsubseteq \neg B_2$ in $\mathcal{T}$ and a constant a such that the assertions $B_1(a)$ and $B_2(a)$ belong to $\mathcal{A}$;
(ii) there exists an assertion $(\mathsf{funct}\ R)$ (respectively, $(\mathsf{funct}\ R^-)$) in $\mathcal{T}$ and three constants a, b, c such that both $R(a, b)$ and $R(a, c)$ (resp., $R(b, a)$ and $R(c, a)$) belong to $\mathcal{A}$.

Informally, condition (i) corresponds to checking whether $\mathcal{A}$ explicitly contradicts some NI in $\mathcal{T}$, and condition (ii) corresponds to check whether $\mathcal{A}$ violates some functionality assertion in $\mathcal{T}$. If one of the above conditions holds, then the algorithm returns *false* (i.e., $\mathcal{K}$ is not satisfiable); otherwise, the algorithm returns *true*.

Notably, the algorithm verifies such conditions by posing to DB($\mathcal{A}$) suitable conjunctive queries expressed in SQL. For instance, condition (i) holds for a given NI $B_1 \sqsubseteq \neg B_2$ iff the query $q(x) \leftarrow tab_{B_1}(x), tab_{B_2}(x)$ has a non-empty

answer in $\mathsf{DB}(\mathcal{A})$, while condition (ii) holds for (funct R) iff the query $q(x) \leftarrow tab_R(x,y), tab_R(x,z), y \neq z$ has a non-empty answer in $\mathsf{DB}(\mathcal{A})$, where $\neq$ is the "not equal" predicate of SQL. Notice that the algorithm does not consider the PIs occurring in $\mathcal{T}$ during its execution. Indeed, we will show that PIs do not affect the consistency of a *DL-Lite* KB, if the TBox is normalized.

Query reformulation Query reformulation is at the heart of our query answering method. Given the limited expressive power of *DL-Lite* TBoxes, it might seem that in order to answer a query q over a KB $\mathcal{K}$, we could simply build a finite first-order structure on the basis of $\mathcal{K}$, and then evaluate the query as an expression over this first-order structure. Actually, it is possible to show that this is not the case. In particular, it can be shown that, in general, given a KB $\mathcal{K}$, there exists no finite structure $\mathcal{S}$ such that, for every conjunctive query q, the set of answers to q over $\mathcal{K}$ is the result of evaluating q over $\mathcal{S}$. This property demonstrates that answering queries in *DL-Lite* goes beyond both propositional logic and relational databases. The basic idea of our method is to reformulate the query taking into account the TBox: in particular, given a query q over $\mathcal{K}$, we compile the assertions of the TBox into the query itself, thus obtaining a new query q'. Such a new query q' is then evaluated over the ABox of $\mathcal{K}$, as if the ABox were a simple relational database. Since the size of q' does not depend on the ABox, the data complexity of the whole query answering algorithm is polynomial. In the following, we illustrate our approach from a technical point of view.

We say that an argument of an atom in a query is *bound* if it corresponds to either a distinguished variable or a shared variable, i.e., a variable occurring at least twice in the query body, or a constant, while we say that it is *unbound* if it corresponds to a non-distinguished non-shared variable (as usual, we use the symbol _ to represent non-distinguished non-shared variables). Notice that, an atom of the form $\exists R(x)$ (resp. $\exists R^-(x)$) has the same meaning as $R(x,\_)$ (resp. $R(\_,x)$). For ease of exposition, in the following we will use the latter form only.

A PI I *is applicable to an atom* $B(x)$, if I has B in its right-hand side, and I *is applicable to an atom* $R(x_1,x_2)$, if either (i) $x_2 = \_$ and the right-hand side of I is $\exists R$, or (ii) $x_1 = \_$ and the right-hand side of I is $\exists R^-$. Roughly speaking, an inclusion I is applicable to an atom g if all bound arguments of g are propagated by I. Obviously, since all PIs in the TBox $\mathcal{T}$ are unary, they are never applicable to atoms with two bound arguments.

We indicate with $gr(g,I)$ the atom obtained from the atom g by applying the inclusion I, i.e., if $g = B_1(x)$ (resp., $g = R_1(x,\_)$ or $g = R_1(\_,x)$) and $I = B_2 \sqsubseteq B_1$ (resp., $I = B_2 \sqsubseteq \exists R_1$ or $I = B_2 \sqsubseteq \exists R_1^-$), we have:

- $gr(g,I) = R_2(x,\_)$, if $B_2 = \exists R_2$;
- $gr(g,I) = R_2(\_,x)$, if $B_2 = \exists R_2^-$;
- $gr(g,I) = A(x)$, if $B_2 = A$, where A is a basic concept.

We are now ready to define the algorithm PerfectRef.

Algorithm PerfectRef($q, \mathcal{T}$)
Input: conjunctive query q, *DL-Lite* TBox $\mathcal{T}$
Output: set of conjunctive queries P
$P := \{q\}$;
repeat
  $P' := P$;
  for each $q \in P'$ **do**
  (a) **for each** g in q **do**
    for each PI I in $\mathcal{T}$ **do**
      if I is applicable to g
      then $P := P \cup \{\, q[g/gr(g,I)] \,\}$
  (b) **for each** g_1, g_2 in q **do**
    if g_1 and g_2 unify
    then $P := P \cup \{\tau(\mathit{reduce}(q, g_1, g_2))\}$;
until $P' = P$;
return P

In the algorithm, $q[g/g']$ denotes the query obtained from q by replacing the atom g with a new atom g'.

Informally, the algorithm first reformulates the atoms of each query $q \in P'$, and produces a new query for each atom reformulation (step (a)). Roughly speaking, PIs are used as rewriting rules, applied from right to left, that allow to compile away in the reformulation the knowledge of $\mathcal{T}$ that is relevant for answering q.

At step (b), for each pair of atoms g_1, g_2 that unify, the algorithm computes the query $q' = \mathit{reduce}(q, g_1, g_2)$, by applying to q the *most general unifier* between g_1 and g_2. Due to the unification, variables that were bound in q may become unbound in q'. Hence, PIs that were not applicable to atoms of q, may become applicable to atoms of q' (in the next executions of step (a)). Function τ applied to q' replaces with _ each unbound variable in q'.

It can be shown that the algorithm always terminates, since the maximum number of atoms in the body of a generated query is equal to the length of the initial query, and the number of different atoms that can be generated by the algorithm is polynomial in the size of the input.

Example 1 (contd.). Let us analyze PerfectRef($q, \mathcal{T}$), where $q(x) \leftarrow \mathit{TeachesTo}(x,y), \mathit{HasTutor}(y,\_)$. At the first execution of step (a), the algorithm inserts in P the new query $q(x) \leftarrow \mathit{TeachesTo}(x,y), \mathit{Student}(y)$, by applying to the atom $\mathit{HasTutor}(y,\_)$ the PI $\mathit{Student} \sqsubseteq \exists \mathit{HasTutor}$. Then, at a second execution of step (a), the query $q(x) \leftarrow \mathit{TeachesTo}(x,y), \mathit{TeachesTo}(\_,y)$ is added to P, according to application of the PI $\exists \mathit{TeachesTo}^- \sqsubseteq \mathit{Student}$ to the atom $\mathit{Student}(y)$. Since the two atoms of the second query unify, step (b) of the algorithm inserts the query $q(x) \leftarrow \mathit{TeachesTo}(x,\_)$ into P. At a next iteration, step (a) produces the query $q(x) \leftarrow \mathit{Professor}(x)$, by applying $\mathit{Professor} \sqsubseteq \exists \mathit{TeachesTo}$ to $\mathit{TeachesTo}(x,\_)$, and then, at a further execution of step (a), it generates the query $q(x) \leftarrow \mathit{HasTutor}(\_,x)$ by applying $\exists \mathit{HasTutor}^- \sqsubseteq \mathit{Professor}$ to $\mathit{Professor}(x)$. The set constituted by the above five queries and the original query q is then returned by the algorithm. ∎

Query evaluation In order to compute the answers to q over the KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, we need to evaluate the set of conjunctive queries P produced by the algorithm PerfectRef over the ABox $\mathcal{A}$. Obviously, in doing so we want to exploit the relational database $\mathsf{DB}(\mathcal{A})$. To this aim, we need to transform each query q in P into an SQL query expressed over $\mathsf{DB}(\mathcal{A})$. The transformation (which we omit for lack of space) is conceptually very simple. The only non-trivial

case concerns binary atoms with unbound terms: for an atom of the form $R(\_, x)$, we introduce a view predicate that represents the union of $tab_R[2]$ with $tab_{\exists R^-}$, where $tab_R[2]$ indicates projection of tab_R on its second column (similarly for $R(x, \_)$). All SQL queries obtained from P, together with the views introduced in the transformation, denoted by $\mathsf{SQL}(P)$, can be easily dispatched to an SQL query engine and evaluated over $\mathsf{DB}(\mathcal{A})$.

Below we define the algorithm Answer that, given a satisfiable KB $\mathcal{K}$ and a query q, computes $ans(q, \mathcal{K})$[6]. In the algorithm, $\mathsf{Eval}(Q, D)$ denotes the evaluation of the SQL query Q over the database D.

Algorithm Answer$(q, \mathcal{K})$
Input: CQ q, *DL-Lite* KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$
Output: $ans(q, \mathcal{K})$
$\mathcal{K}$:= Normalize($\mathcal{K}$);
return $\mathsf{Eval}(\mathsf{SQL}(\mathsf{PerfectRef}(q, \mathcal{T})), \mathsf{DB}(\mathcal{A}))$

Example 1 (contd.). Since our ABox $\mathcal{A}$ contains only the assertion $HasTutor(\mathsf{John}, \mathsf{Mary})$, it is trivial to establish satisfiability of $\mathcal{K}$ (which can be done by means of the algorithm Consistent). Then, by executing Answer$(q, \mathcal{K})$, we first obtain Normalize$(\mathcal{K})$, which is computed by adding to $\mathcal{T}$ all NIs implied by $\mathcal{T}$, i.e.,:

$$\exists TeachesTo^- \sqsubseteq \neg Professor \quad \exists HasTutor^- \sqsubseteq \neg Student.$$

Then, $\mathsf{Eval}(\mathsf{SQL}(\mathsf{PerfectRef}(q, \mathcal{T})), \mathsf{DB}(\mathcal{A}))$ returns the set $\{\mathsf{Mary}\}$. In particular, Mary is returned by the evaluation of the SQL transformation of the query $q(x) \leftarrow HasTutor(\_, x)$. ∎

Correctness We now prove correctness of the above described query answering technique. To this aim, we use a chase-like technique [2]. Given a normalized KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, we call *chase of* $\mathcal{K}$ (denoted by *chase*$(\mathcal{K})$) the (possibly infinite) ABox obtained starting from $\mathcal{A}$ and closing it with respect to the following chase rules: (i) if $B_1(a) \in$ *chase*$(\mathcal{K})$ and $B_1 \sqsubseteq B_2 \in \mathcal{T}$, then add $B_2(a)$ to *chase*$(\mathcal{K})$; (ii) if $\exists R(a) \in$ *chase*$(\mathcal{K})$ (respectively, $\exists R^-(a) \in$ *chase*$(\mathcal{K})$) and there exists no individual b such that $R(a, b) \in$ *chase*$(\mathcal{K})$ (resp., $R(b, a) \in$ *chase*$(\mathcal{K})$), then add the assertions $R(a, n)$ and $\exists R^-(n)$ (resp., $R(n, a)$ and $\exists R(n)$) to *chase*$(\mathcal{K})$, where n is a new constant of Δ not already occurring in *chase*$(\mathcal{K})$.

Intuitively, the correctness of our query processing technique is based on a crucial property of *chase*$(\mathcal{K})$: if $\mathcal{K}$ is satisfiable, then *chase*$(\mathcal{K})$ is a representative of all models of $\mathcal{K}$. This property implies that query answering can be in principle done by evaluating the query over *chase*$(\mathcal{K})$ seen as a database. However, since *chase*$(\mathcal{K})$ is in general infinite, we obviously avoid the construction of the chase. Rather, as we said before, we are able to compile the TBox into the query, thus simulating the evaluation of the query over the (in general infinite) chase by evaluating a finite reformulation of the query over the initial ABox.

We first establish correctness of the technique for deciding satisfiability of a *DL-Lite* KB.

[6]Notice that, if $\mathcal{K}$ is unsatisfiable, query answering is meaningless, since every tuple is in the answer to every query.

Theorem 2 *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a normalized DL-Lite KB. $\mathcal{K}$ is satisfiable iff the algorithm Consistent returns true.*

We now establish correctness of the algoritm Answer under the assumption that the KB is satisfiable.

Theorem 3 *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a satisfiable DL-Lite KB, let q be a CQ, and let $\vec{c}$ be a tuple of constants from Δ. Then, $\vec{c} \in ans(q, \mathcal{K})$ iff $\vec{c} \in$ Answer$(q, \mathcal{K})$.*

Complexity First, we analyze complexity of KB satisfiability in *DL-Lite*.

Theorem 4 *Satisfiability of a DL-Lite KB $\mathcal{K}$ can be decided in time polynomial in the size of $\mathcal{K}$.*

Proof (sketch). The proof immediately follows from the following facts: (i) the algorithm Normalize runs in time polynomial in the size of $\mathcal{K}$; (ii) the algorithm Consistent is correct; (iii) the algorithm Consistent runs in time polynomial in the size of the input. □

Then, from correctness of the algorithm Answer, we are immediately able to characterize complexity of conjunctive query answering in *DL-Lite* w.r.t. data complexity.

Theorem 5 *Conjunctive query answering in DL-Lite is in PTIME in data complexity.*

We are also able to characterize the combined complexity (i.e., the complexity w.r.t. the size of $\mathcal{K}$ and q) of conjunctive query answering in *DL-Lite*.

Theorem 6 *Conjunctive query answering in DL-Lite is NP-complete in combined complexity.*

Proof (sketch). Membership in NP is a consequence of the fact that, given any *DL-Lite* KB $\mathcal{K}$, if $\vec{c} \in ans(q, \mathcal{K})$, then it is possible to nondeterministically construct a *polynomial* fragment of *chase*$(\mathcal{K})$ which contains an image of $q(\vec{c})$. NP-hardness follows from NP-hardness of conjunctive query evaluation over relational databases. □

Finally, since in *DL-Lite* it is possibile to polynomially reduce containment between CQs to query answering, from the above results it follows that containment of conjunctive queries in *DL-Lite* is NP-complete.

Summarizing, the above results show a very nice computational behavior of queries in *DL-Lite*: reasoning in *DL-Lite* is computationally no worse than standard conjunctive query answering (and containment) in relational databases.

Discussion and related work

DL-Lite is a fragment of expressive DLs with assertions and inverses studied in the 90's (see [4] for an overview), which are at the base of current ontology languages such as OWL, and for which optimized automated reasoning systems such as Fact and Racer have been developed. Indeed, one could use, off-the-shelf, a system like Racer to perform KB satisfiability, instance checking (of concepts), and logical implication of inclusion assertions in *DL-Lite*. Also, reasoning with conjunctive queries in these DLs has been studied (see e.g. [11]), although not yet implemented in systems. Unfortunately, the reasoning procedures for these DLs are all EXPTIME-hard, and more importantly they are not tailored

towards obtaining tight complexity bounds with respect to data complexity. Conjunctive queries combined with DLs were also considered in [16, 13], but again data complexity was not the main concern.

There has been a lot of work in DLs on the boundary between polynomial and exponential reasoning. This work first concentrated on DLs without the TBox component of the KB, and led to the development of simple DLs, such as $\mathcal{ALN}$, that admit polynomial instance checking. However, for minor variants of $\mathcal{ALN}$, such as $\mathcal{ALE}$ (where we introduce qualified existential and drop number restrictions), $\mathcal{FLE}^-$ (where we additionally drop negated atomic concept), and $\mathcal{ALU}$ (where we introduce union and drop number restrictions), instance checking, and therefore conjunctive query answering, is coNP-complete in data complexity [12]. Indeed, the argument used in the proof of coNP-hardness of $\mathcal{ALE}$, $\mathcal{FLE}^-$, and $\mathcal{ALU}$ in [12], immediately implies the following theorem.

Theorem 7 *Answering conjunctive queries is coNP-hard in data complexity (even in KBs with empty TBoxes), if we extend DL-Lite with one of the following features: (1) either $\forall R.A$ or $\neg A$ can appear in left-hand sides of inclusion assertions; (2) either $\forall R.A$ or $\neg A$ can appear as atoms in the query; (3) union of concepts can appear in the right-hand side of inclusion assertions.*

If we allow for cyclic inclusion assertions in the KB, then even subsumption in CLASSIC and $\mathcal{ALN}$ becomes intractable [9][7]. Observe that *DL-Lite* does allow for cyclic assertions without falling into intractability. Indeed, we can enforce the cyclic propagation of the existence of an R-successor using the two *DL-Lite* inclusion assertions $A \sqsubseteq \exists R$, $\exists R^- \sqsubseteq A$. The constraint imposed on a model is similar to the one imposed by the $\mathcal{ALN}$ cyclic assertion $A \sqsubseteq \exists R \sqcap \forall R.A$, though stronger, since it additionally enforces the second component of R to be typed by A. In order to keep tractability even in the presence of cycles, *DL-Lite* imposes restrictions on the use of the $\forall R.C$ construct, which, if used together with inclusion assertions, immediately would lead to intractability [9].

Our work is also tightly related to work in databases on implication of integrity constraints (ICs) [2] and on query answering in the presence of ICs under an open world semantics (see, e.g., [8, 3, 14, 7]). Rephrased as ICs, *DL-Lite* TBoxes allow for expressing special forms of inclusion dependencies (i.e., ISA, role typing, and participation constraints), multiple keys on relations (i.e., functionality restrictions), and exclusion dependencies (i.e., disjointness and non-participation constraints)[8]. The results that we report here show that *DL-Lite* inclusion assertions form one of the largest class of ICs for which query answering remains polynomial.

Conclusions

We have described *DL-Lite*, a new DL specifically tailored to capture conceptual data models and basic ontology languages, while keeping the worst-case complexity of sound and complete reasoning tractable.

In this paper we focused on binary roles only, but it is possible to extend our reasoning techniques to n-ary relations without loosing their nice computational properties. We are working on other interesting extensions to *DL-Lite*, such as the introduction of subset constraints on roles. The results of [10] imply that finding an adaptation of our query answering technique is going to be a hard problem.

Acknowledgments This research has been partially supported by the Projects INFOMIX (IST-2001-33570) and SEWASIE (IST-2001-34825) funded by the EU.

[7] Note that a TBox with only acyclic inclusion assertions can always be transformed into an empty TBox.

[8] Notice that this combination of ICs has only been studied in [7], but under a different semantics wrt the one adopted in DLs.

References

[1] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Proc. of PODS'98*, pages 254–265, 1998.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., 1995.

[3] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proc. of PODS'99*, pages 68–79, 1999.

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

[5] A. Borgida and R. J. Brachman. Conceptual modeling with description logics. In Baader et al. [4], chapter 10, pages 349–372.

[6] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick. CLASSIC: A structural data model for objects. In *Proc. of ACM SIGMOD*, pages 59–67, 1989.

[7] L. Bravo and L. Bertossi. Logic programming for consistently querying data integration systems. In *Proc. of IJCAI 2003*, pages 10–15, 2003.

[8] A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. of PODS 2003*, pages 260–271, 2003.

[9] D. Calvanese. Reasoning with inclusion axioms in description logics: Algorithms and complexity. In *Proc. of ECAI'96*, pages 303–307. John Wiley & Sons, 1996.

[10] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. What to ask to a peer: Ontology-based query reformulation. In *Proc. of KR 2004*, pages 469–478, 2004.

[11] D. Calvanese, G. De Giacomo, and M. Lenzerini. Answering queries using views over description logics knowledge bases. In *Proc. of AAAI 2000*, pages 386–391, 2000.

[12] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Deduction in concept languages: From subsumption to instance checking. *J. of Log. and Comp.*, 4(4):423–452, 1994.

[13] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. $\mathcal{AL}$-log: Integrating Datalog and description logics. *J. of Intelligent Information Systems*, 10(3):227–252, 1998.

[14] O. M. Duschka, M. R. Genesereth, and A. Y. Levy. Recursive query plans for data integration. *J. of Logic Programming*, 43(1):49–73, 2000.

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

[16] A. Y. Levy and M.-C. Rousset. Combining Horn rules and description logics in CARIN. *Artificial Intelligence*, 104(1–2):165–209, 1998.

[17] M. Y. Vardi. The complexity of relational query languages. In *Proc. of STOC'82*, pages 137–146, 1982.