Efficiently Managing Data Intensive 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 Bozen-Bolzano
Piazza Domenicani 3
I-39100 Bolzano, Italy
calvanese@inf.unibz.it

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

Abstract. The idea of using ontologies as a conceptual view over data repositories is becoming more and more popular. In these contexts, data are typically very large (much larger than the intentional level of the ontologies), and query answering becomes the basic reasoning services. In these contexts query answering should be very efficient on the data, and currently the only technology that is available to deal with large amounts of data is the one provided by relational data management systems (RDBMS). In this paper we advocate that for such contexts a suitable fragment of OWL-DL should be devised. Such a fragment must allow forms of query answering that exploit RDBMS when reasoning on the data, while it must include the main modeling features of conceptual models like UML class diagrams and ER diagrams. In particular it must include cyclic assertions, ISA on concepts, inverses of roles, role typing, mandatory participation to roles, and functional restrictions on roles. Also the query language should go beyond the expressive capabilities of concept expressions in description logics, and include at least conjunctive queries (corresponding to the select-project-join fragment of SQL). We discuss this issues by exhibiting a fragment of OWL-DL that includes all such features, namely *DL-Lite*, and showing that such a fragment is essentially maximal.

1 Introduction

The idea of using ontologies as a conceptual view over data repositories is becoming more and more popular. For example, in Enterprise Application Integration Systems, Data Integration Systems [16], and the Semantic Web [13], data become instances of concepts in ontologies. In these contexts, data are typically very large and dominate the intensional level of the ontologies. Hence, when measuring the computational complexity of reasoning, the most important parameter is the size of the data, i.e., one is interested in *data complexity* [20]. While in all the above mentioned contexts one could still accept reasoning that is exponential on the intensional part, it is mandatory that reasoning is polynomial (actually less – see later) in the data. A second fundamental requirement is the possibility to answer queries over an ontology that are more complex than the simple queries (i.e., concepts and roles) usually considered in Description Logics (DLs) research.

Traditionally, research carried out in DLs has not paid much attention to data complexity (see Section 4 for a detailed discussion), and only recently efficient management of large amounts of data has become a primary concern in ontology reasoning systems [14, 10]. Unfortunately, research on the trade-off between expressive power and computational complexity of reasoning has shown that many DLs with efficient, i.e., worst-case polynomial time, reasoning algorithms lack the modeling power required for capturing conceptual models (such as UML class diagrams and Entity-Relationship diagrams) and basic ontology languages. On the other hand, whenever the complexity of reasoning is exponential in the size of the instances (as for example for OWL-DL, in Racer[1], and in [9]), there is little hope for effective instance management. Indeed, the only technology that is currently available to deal with complex queries over large amounts of data is the one provided by relational data management systems (RDBMS), and it cannot be directly exploited in these cases.

In this paper we advocate that for those contexts where ontologies are used to access large amounts of data, a suitable fragment of OWL-DL should be devised, specifically tailored to capture conceptual modeling constructs, while keeping query answering efficient. Specifically, efficiency of query answering should be achieved by delegating data storage and query answering to an RDBMS. The fragment should include the main modeling features of conceptual models, which are also at the base of most ontology languages. These features include cyclic assertions, ISA on concepts, inverses on roles, role typing, mandatory participation to roles, and functional restrictions of roles. Also, the query language should go beyond the expressive capabilities of concept expressions in DLs, and include at least conjunctive queries (corresponding to the select-project-join fragment of SQL).

We present a DL, called *DL-Lite*, that exhibits all the above characteristics [8]. The distinguishing features of *DL-Lite* are that the extensional component of a knowledge base, the ABox, is maintained by an RDBMS in secondary storage, and that query answering can be performed as a two step process: in the first step, a query posed over the knowledge base is reformulated, taking into account the intensional component (the TBox) only, obtaining a union of conjunctive queries; in the second step such a union is directly evaluated over the ABox, and the evaluation can be carried out by an SQL engine, taking advantage of well established query optimization strategies. Since the first step does not depend on the data, and the second step is the evaluation of a relational query over a databases, the whole query answering process is in LOGSPACE in the data [1].

We show also that *DL-Lite* is essentially the maximal fragment exhibiting such a desirable property, and allowing one to delegate query evaluation to a relational engine [8]. Indeed, even slight extensions of *DL-Lite* make query answering (actually already instance checking, i.e., answering atomic queries) at least NLOGSPACE in data complexity, ruling out the possibility that query evaluation could be performed by a relational engine.

2 *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.

[1] `http://www.sts.tu-harburg.de/~r.f.moeller/racer/`

In *DL-Lite*[2], concepts and roles are defined as follows:

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

where A denotes an atomic concept and P denotes an atomic role; R denotes a (generic) role, which can either be an atomic role or its *inverse*; C (possibly with subscript) denotes a (generic) concept that can be either an atomic concept, the empty concept $\bot$, a concept of the form $\exists R$, i.e., the standard DL construct of unqualified existential quantification on roles, or the conjunction of two concepts. Note that we allow for a limited form of negation through $\bot$ (sufficient to capture disjointness of concepts), but we do not allow for disjunction.

A *DL-Lite* knowledge base (KB) $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is constituted by two components: a TBox $\mathcal{T}$, used to represent intensional knowledge, and an ABox $\mathcal{A}$, used to represent extensional information. *DL-Lite TBox* assertions are of the form

$$\begin{array}{ll} C_1 \sqsubseteq C_2 & \textit{inclusion assertion} \\ (\mathsf{funct}\ R) & \textit{functionality assertion} \end{array}$$

An inclusion assertion expresses that a concept C_1 is subsumed by a concept C_2, while a functionality assertion expresses the (global) functionality of a role (atomic or inverse).

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

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

where a and b are constants. These assertions state respectively that the object denoted by a is an instance of the concept C, 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 $q(\boldsymbol{x}) \leftarrow \exists \boldsymbol{y}.conj(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x}$ are the so-called *distinguished variables*, $\boldsymbol{y}$ are existentially quantified variables called the *non-distinguished* variables, and $conj(\boldsymbol{x}, \boldsymbol{y})$ is a conjunction of atoms of the form $C(z)$, or $R(z_1, z_2)$, where C and R are respectively a concept and a role in $\mathcal{K}$, and z, z_1, z_2 are constants of a fixed infinite *domain* Δ (see later) or variables in $\boldsymbol{x}$ or $\boldsymbol{y}$.

The semantics of *DL-Lite* is given in terms of interpretations over a fixed infinite *domain* Δ[3]. We assume to have one constant for each object, denoting exactly that object. In other words, we have *standard names* [17], and we will not distinguish between the alphabet of constants and Δ.

[2] The variant of *DL-Lite* presented here is a slight extension of the language presented in [8, 7], since it allows for conjunction of concepts in the left-hand side of inclusion assertions.

[3] The assumption of having a fixed interpretation domain is done for convenience in the definition of query answering (see later).

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:

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

An interpretation $\mathcal{I}$ is a *model* of an inclusion assertion $C_1 \sqsubseteq C_2$ if and only if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$; $\mathcal{I}$ is a model of a functionality assertion $(\mathsf{funct}\ R)$ if $(c, c') \in R^{\mathcal{I}}$ and $(c, c'') \in R^{\mathcal{I}}$ implies $c' = c''$; $\mathcal{I}$ is a model of a membership assertion $C(a)$ (resp., $R(a, b)$) if $a \in C^{\mathcal{I}}$ (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 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 $q(\boldsymbol{x}) \leftarrow \exists \boldsymbol{y}. conj(\boldsymbol{x}, \boldsymbol{y})$ is interpreted in $\mathcal{I}$ as the set $q^{\mathcal{I}}$ of tuples $\boldsymbol{c} \in \Delta \times \cdots \times \Delta$ such that, when we substitute the variables $\boldsymbol{x}$ with the constants $\boldsymbol{c}$, the formula $\exists \boldsymbol{y}. conj(\boldsymbol{x}, \boldsymbol{y})$ evaluates to true in $\mathcal{I}$.

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

– *Query answering*: given a query q with distinguished variables $\boldsymbol{x}$ and a KB $\mathcal{K}$, return the set $ans(q, \mathcal{K})$ of tuples $\boldsymbol{c}$ of constants of $\mathcal{K}$ such that $\boldsymbol{c} \in q^{\mathcal{I}}$, for every model $\mathcal{I}$ of $\mathcal{K}$. 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 $q_1^{\mathcal{I}} \subseteq q_2^{\mathcal{I}}$, for every model $\mathcal{I}$ of $\mathcal{K}$. Note that this task generalizes *logical implication* of inclusion assertions in DLs.
– *KB satisfiability*: verify whether a KB is satisfiable.

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, such as Entity-Relationship diagrams and UML class diagrams. In particular, *DL-Lite* assertions allow us to specify: *ISA*, e.g., stating that a concept A_1 is subsumed by a concept A_2, using $A_1 \sqsubseteq A_2$; *disjointness*, e.g., between concepts A_1 and A_2, using $A_1 \sqcap A_2 \sqsubseteq \bot$; *role-typing*, e.g., stating that the first (resp., second) component of the role P is an instance of A, using $\exists P \sqsubseteq A$ (resp., $\exists P^- \sqsubseteq A$); *participation constraints*, e.g., stating that all instances of a concept A participate to a role P as the first (resp., second) component, using $A \sqsubseteq \exists P$ (resp., $A \sqsubseteq \exists P^-$); *non-participation constraints*, using $A \sqcap \exists R \sqsubseteq \bot$; *functionality restrictions* on roles, using $(\mathsf{funct}\ R)$.

Observe that *DL-Lite* does allow for cyclic assertions without falling into intractability. Indeed, we can enforce the cyclic propagation of the existence of a P-successor using the two *DL-Lite* inclusion assertions $A \sqsubseteq \exists P$ and $\exists P^- \sqsubseteq A$. The constraint imposed on a model is similar to the one imposed by the $\mathcal{ALN}$ cyclic assertion $A \sqsubseteq$

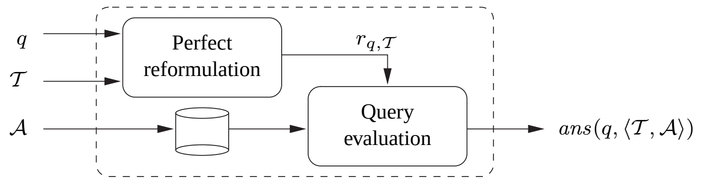

Fig. 1. Query aswering via query evaluation

$\exists P \sqcap \forall P.A$, though stronger, since it additionally enforces the second component of P 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 of TBox reasoning [6] and of query answering (cf. Table 1).

Finally, notice that *DL-Lite* is a strict subset of OWL-Lite, the least expressive variant of OWL[4], which presents some constructs (e.g., some kinds of role restrictions) that cannot be expressed in *DL-Lite*, and that make reasoning in OWL-Lite non-tractable in general.

3 Reasoning in *DL-Lite*

We discuss now reasoning in *DL-Lite*, and concentrate on the basic reasoning task in the context of using ontologies to access large data repositories, namely that of answering (conjunctive) queries over a *DL-Lite* knowledge base. The other forms of reasoning can be reduced to query answering [8]. For example, to check whether $\mathcal{K}$ is unsatisfiable, we can simply add the inclusion $A_1 \sqcap A_2 \sqsubseteq \bot$ to the TBox and the assertion $A_1(a)$ to the ABox (where A_1, A_2 are new atomic concepts and a is new constant), and check whether a is in the answer to the query $q(x) \leftarrow A_2(x)$. Similarly, to check whether $\mathcal{K}$ implies $A \sqsubseteq C$, we can simply add the assertion $A(a)$ to the Abox (where a is new constant), and check whether a is in the answer to the query $q(x) \leftarrow C'(x)$, where C' is the conjunction of atoms corresponding to the concept C.

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 itself 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 *DL-Lite* 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 itself over $\mathcal{S}$ (see [5]). This property demonstrates that answering queries in *DL-Lite* goes beyond both propositional logic and relational databases.

Instead, in order to exploit query evaluation for query answering, and also properly take into account that the size of the ABox (i.e., the data) largely dominates the size of the TBox, we consider the query answering process as divided in two steps (cf. Figure 1):

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

1. First, considering the TBox $\mathcal{T}$ only, the user query q is reformulated into a new query $r_{q,\mathcal{T}}$ (expressed in a suitable query language $\mathcal{L}_Q$).
2. Then, the reformulated query $r_{q,\mathcal{T}}$ is evaluated over the ABox $\mathcal{A}$ only (considered as a first-order structure, i.e., a database), producing the requested answer $ans(q, \langle\mathcal{T}, \mathcal{A}\rangle)$.

Notice that, in principle, such a two steps query answering process is always possible for arbitrary TBoxes and user queries, provided we do not impose any restriction on the query language $\mathcal{L}_Q$ in which the reformulation $r_{q,\mathcal{T}}$ is expressed. Indeed, in order to ensure that the evaluation of $r_{q,\mathcal{T}}$ over the ABox $\mathcal{A}$ produces the correct answer $ans(q, \langle\mathcal{T}, \mathcal{A}\rangle)$ (i.e., that $r_{q,\mathcal{T}}$ is a *perfect reformulation* of q given $\mathcal{T}$), we may have to allow for $\mathcal{L}_Q$ to be completely general. In other words, we may need to be able to specify in $r_{q,\mathcal{T}}$ an arbitrary Turing Machine computation, possibly one that performs full inferences over the TBox and the query. However, if we pose no restriction on $\mathcal{L}_Q$, and hence on the form of $\mathcal{T}$ and q, we have no guarantee that the evaluation of $r_{q,\mathcal{T}}$ over the ABox $\mathcal{A}$ can be performed efficiently, and hence that the separation between query reformulation (using the TBox only) and query evaluation (over the ABox only) makes sense from a computational complexity point of view.

As shown in [8, 7], one of the distinguishing features of *DL-Lite* is that the above described two steps query answering process makes sense, and allows us to be efficient in the size of the data. Indeed, the perfect reformulation $r_{q,\mathcal{T}}$ of a conjunctive query q over a *DL-Lite* KB $\mathcal{K} = \langle\mathcal{T}, \mathcal{A}\rangle$ can be expressed as a union of conjunctive queries, i.e., a set of select-project-join SQL queries, and hence the query evaluation step can be performed in LOGSPACE in the size of the ABox $\mathcal{A}$ [1]. Since the size of $r_{q,\mathcal{T}}$ does not depend on $\mathcal{A}$, the data complexity (i.e., the complexity measured as a function of the size of the ABox only) of the whole query answering algorithm is LOGSPACE.

Moreover, by storing the ABox under the control of an RDBMS, which can manage effectively large numbers (i.e., millions) of objects in the knowledge base, we can delegate the query evaluation step to an SQL database engine. More precisely, we can construct a relational database that faithfully represents an ABox $\mathcal{A}$ as follows. First of all, we assume that $\mathcal{A}$ does not contain conjunction and $\bot$. If this is not the case, we can easily pre-process it and bring it in such a form in linear time (actually, in LOGSPACE in the number of objects in the ABox). Let further $\mathcal{A}'$ be the ABox obtained from $\mathcal{A}$ by adding for each assertion $R(a, b) \in \mathcal{A}$ the implied assertions $\exists R(a)$ and $\exists R^-(b)$. Then, for each concept C in $\mathcal{K}$ that is either atomic or of the form $\exists R$, we define a relational table tab_C of arity 1, such that $\langle a\rangle \in tab_C$ if and only if $C(a) \in \mathcal{A}'$. Similarly, for each atomic role P in $\mathcal{K}$, we define a relational table tab_P of arity 2, such that $\langle a, b\rangle \in tab_P$ if and only if $P(a, b) \in \mathcal{A}$ or $P^-(b, a) \in \mathcal{A}$. Let us call $DB(\mathcal{A})$ the resulting database. The fact that the ABox $\mathcal{A}$ is managed in secondary storage by an RDBMS, together with the fact that in *DL-Lite* the perfect reformulation of a select-project-join SQL query can be expressed as a set of select-project-join SQL queries that can be directly evaluated over $DB(\mathcal{A})$, allows us to completely delegate the query evaluation process to the SQL engine of the RDBMS, and to take advantage of well established query optimization strategies.

We have developed a prototype tool, called QUONTO [2] (see Figure 2), that implements the *DL-Lite* query answering algorithm and delegates to an RDBMS the ABox

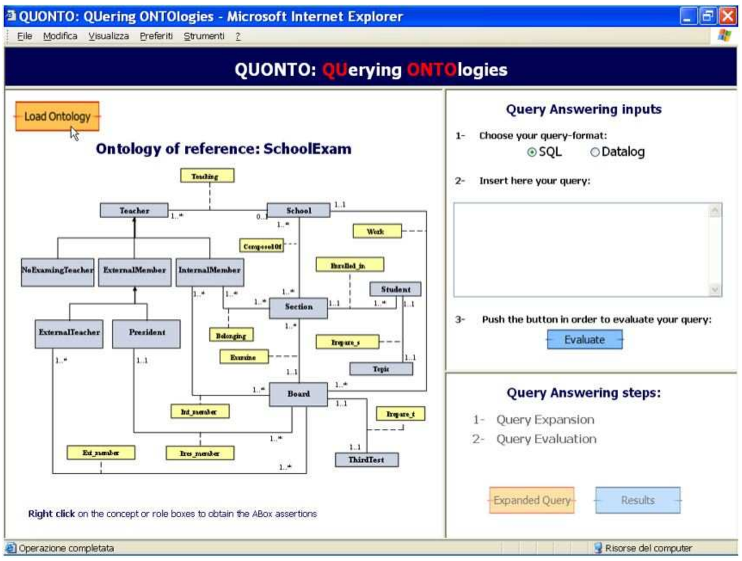

Fig. 2. Screenshot of the QUONTO query answering tool

storage and the query evaluation step. QUONTO is able to answer queries over ABoxes containing millions of assertions, and the limitations actually depend on the underlying DBMS engine (currently we use MySQL).

Notice that, as soon as we extend the expressive power of *DL-Lite* even slightly, we lose the possibility of delegating query evaluation to an RDBMS. Indeed, Table 1, drawn from [7], shows bounds on the data complexity of query answering for various DLs obtained from *DL-Lite* by adding various constructs (and possibly removing some). To give a more precise account of the complexity, we distinguish between the constructs allowed in concepts on the left-hand side of inclusion assertions (denoted by B), and those allowed in concepts on the right-hand side of inclusion assertions (denoted by C). The language $\mathcal{L}_1$ is *DL-Lite*, and $\mathcal{L}_2$ is obtained from *DL-Lite* by allowing for qualified existential quantification in C while forbidding the use of functionality assertions. Similarly to *DL-Lite*, query answering in $\mathcal{L}_2$ can be performed in LOGSPACE via query reformulation. Instead, in languages $\mathcal{L}_3$, $\mathcal{L}_4$, $\mathcal{L}_5$, one can encode reachability in directed graphs (essentially through the use of qualified existentials in B). Hence query answering (actually, already instance checking) becomes NLOGSPACE-hard in data-complexity. By further allowing for the use of conjunction in B, one can encode Path System Accessibility (a non-linear form of reachability), and instance checking becomes PTIME-hard ($\mathcal{L}_6$, $\mathcal{L}_7$, $\mathcal{L}_8$). Finally, by allowing to denote two concepts that together cover the whole domain, conjunctive query answering becomes even coNP-hard in data-complexity ($\mathcal{L}_9$, $\mathcal{L}_{10}$, $\mathcal{L}_{11}$).

The fact that the data-complexity goes beyond LOGSPACE, means actually that query answering (resp., instance checking) requires more powerful engines than those available in standard relational database technology. Essentially NLOGSPACE-hardness

Table 1. Data complexity of query answering in extensions of *DL-Lite*

$\mathcal{L}_i$	B	C	R	$(\mathsf{funct}\ R)$	Complexity
$\mathcal{L}_1$	$A \mid \bot \mid \exists R \mid B_1 \sqcap B_2$	$A \mid \bot \mid \exists R \mid C_1 \sqcap C_2$	$P \mid P^-$	*allowed*	in LOGSPACE
$\mathcal{L}_2$	$A \mid \bot \mid \exists R \mid B_1 \sqcap B_2$	$A \mid \bot \mid \exists R.C \mid C_1 \sqcap C_2$	$P \mid P^-$	*not all.*	in LOGSPACE
$\mathcal{L}_3$	$A \mid \exists P.A$	A	P	*not all.*	NLOGSPACE-hard
$\mathcal{L}_4$	A	$A \mid \forall P.A$	P	*not all.*	NLOGSPACE-hard
$\mathcal{L}_5$	A	$A \mid \exists P.A$	P	*allowed*	NLOGSPACE-hard
$\mathcal{L}_6$	$A \mid \exists P.A \mid A_1 \sqcap A_2$	A	P	*not all.*	PTIME-hard
$\mathcal{L}_7$	$A \mid A_1 \sqcap A_2$	$A \mid \forall P.A$	P	*not all.*	PTIME-hard
$\mathcal{L}_8$	$A \mid A_1 \sqcap A_2$	$A \mid \exists P.A$	P	*allowed*	PTIME-hard
$\mathcal{L}_9$	$A \mid \neg A$	A	P	*not all.*	coNP-hard
$\mathcal{L}_{10}$	A	$A \mid A_1 \sqcup A_2$	P	*not all.*	coNP-hard
$\mathcal{L}_{11}$	$A \mid \forall P.A$	A	P	*not all.*	coNP-hard

Legend: A (possibly with subscript)= atomic concept, P = atomic role,
B (possibly with subscript) = left-hand side of TBox inclusion assertions,
C = right-hand side of TBox inclusion assertions, R = arbitrary role.

means that at least the power to compute transitive closure (i.e., linear recursion) is required, while PTIME-hardness essentially requires the power of Datalog. For the coNP-hard cases a technology based on Disjunctive Datalog could be adopted. An immediate consequence of this fact is that, as soon as we go beyond *DL-Lite*, we lose the possibility of delegating query answering to data management tools and we cannot take advantage of query optimization techniques of current industrial strength RDBMSs.

4 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[5], Racer, and Pellet[6] have been developed. Indeed, one could use, off-the-shelf, a system like Racer or Pellet 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., [9]), 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 bounds with respect to data complexity. Alternative reasoning procedures that allow for clearly isolating data complexity have recently been proposed, how they will work in practice still needs to be understood: a coNP upper bound for data complexity of instance checking in an expressive DL has been shown, and a polynomial fragment has been isolated [15], though it is open whether the technique can be extended to deal efficiently with conjunctive queries; building on the technique in [18], coNP-completeness of answering conjunc-

[5] `http://www.cs.man.ac.uk/~horrocks/FaCT/`
[6] `http://www.mindswap.org/2003/pellet/`

tive queries for an expressive DL with assertions, inverse roles, and number restrictions (that generalize functionality) has been shown [19].

DL-Lite can also capture (the DL-subset of) RDFS[7], except for role hierarchies. In fact, the query answering technique for *DL-Lite* works also for full RDFS extended with participation constraints (i.e., inclusion assertions with $\exists R$ on the right-hand side), and one can show that in this case query answering is indeed LOGSPACE. However, if we further extend RDFS with functionality assertions, it can be shown that query answering becomes NLOGSPACE-hard. Finally, if we move from RDFS to DLP [12], query answering becomes PTIME-hard, since DLP is a superset of $\mathcal{L}_6$ in Table 1.

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), conjunctive query answering is coNP-hard in data complexity [11]. If we allow for cyclic inclusion assertions in the KB, then even subsumption in CLASSIC and $\mathcal{ALN}$ becomes intractable [6][8].

More recently languages equipped with *qualified existential restrictions* but no universal restrictions (even expressed in a disguised way, as in *DL-Lite*, through inverse roles) have been studied. In particular, in [3] it has been shown that for the basic language $\mathcal{EL}$ instance checking is polynomial even in the presence of general (i.e., cyclic) inclusion assertions, while extensions of the language lead easily to intractability (cf. Table 1). Conjunctive query answering in $\mathcal{EL}$ has not been studied yet, however the results in Table 1 show us that such a service is PTIME-hard in data complexity and hence cannot be delegated to a relational DBMS (actually such a lower bound holds already for instance checking).

5 Conclusions

For the management of data intensive ontologies, we have advocated the use of *DL-Lite*, a subset of OWL-Lite that (*i*) is specifically tailored to capture conceptual data models and basic ontology languages, and (*ii*) keeps the worst-case data complexity of sound and complete reasoning in LOGSPACE. This allows one to effectively exploit current industrial strength relational technology for query answering.

We have also argued that by extending *DL-Lite*, we lose this possibility, and in order to allow for the separation of TBox and ABox reasoning, we have to rely on more powerful query evaluation engines, such as one for Datalog. Hence, the possibility of efficient implementations and optimization strategies for query evaluation engines that go beyond relational ones is worth investigating.

[7] `http://www.w3.org/TR/rdf-schema/`

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

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., 1995.
2. A. Acciarri, D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, and R. Rosati. QuOnto: QUerying ONTOlogies. In *Proc. of AAAI 2005*, pages 1670–1671, 2005.
3. F. Baader, S. Brandt, and K. Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proc. of IJCAI 2005*, 2005.
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. Calì, D. Lembo, and R. Rosati. Query rewriting and answering under constraints in data integration systems. In *Proc. of IJCAI 2003*, pages 16–21, 2003.
6. D. Calvanese. Reasoning with inclusion axioms in description logics: Algorithms and complexity. In *Proc. of ECAI'96*, pages 303–307. John Wiley & Sons, 1996.
7. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In *Proc. of DL 2005*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/`, 2005.
8. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. DL-Lite: Tractable description logics for ontologies. In *Proc. of AAAI 2005*, pages 602–607, 2005.
9. 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.
10. C. Chen, V. Haarslev, and J. Wang. LAS: Extending Racer by a Large ABox Store. In *Proc. of DL 2005*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/`, 2005.
11. 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.
12. B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *Proc. of WWW 2003*, pages 48–57, 2003.
13. J. Heflin and J. Hendler. A portrait of the semantic web in action. *IEEE Intelligent Systems*, 16(2):54–59, 2001.
14. I. Horrocks, L. Li, D. Turi, and S. Bechhofer. The Instance Store: DL reasoning with large numbers of individuals. In *Proc. of DL 2004*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-104/`, 2004.
15. U. Hustadt, B. Motik, and U. Sattler. Data complexity of reasoning in very expressive description logics. In *Proc. of IJCAI 2005*, 2005.
16. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of PODS 2002*, pages 233–246, 2002.
17. H. J. Levesque and G. Lakemeyer. *The Logic of Knowledge Bases*. The MIT Press, 2001.
18. A. Y. Levy and M.-C. Rousset. Combining Horn rules and description logics in CARIN. *Artificial Intelligence*, 104(1–2):165–209, 1998.
19. M. M. Ortiz de la Fuente, D. Calvanese, T. Eiter, and E. Franconi. Data complexity of answering conjunctive queries over $\mathcal{SHIQ}$ knowledge bases. Technical report, Fac. of Computer Science, Free Univ. of Bozen-Bolzano, July 2005. Also available as CORR technical report at `http://arxiv.org/abs/cs.LO/0507059/`.
20. M. Y. Vardi. The complexity of relational query languages. In *Proc. of STOC'82*, pages 137–146, 1982.