International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 4 Issue: 3 248 - 252

DL-Lite: Tractable Description Logics for Ontologies: A Survey

M. P. Bidve
Department of Computer Science and Engineering
M.S.Bidve Engg. Collage
Latur, Maharastra, India
manishabidve2@gmail.com

Prof. N. J. Pathan
Department of Computer Science and Engineering
M.S.Bidve Engg. Collage
Latur, Maharastra, India
pathan_nj@rediffmail.com

***Abstract*—** Description Logic, called *DL-Lite*, specially used to capture essential ontology languages, and keeping low difficulty of logic. Here logic means computing subsumption between concepts, and checking satisfiability of the whole knowledge base, as well as answer complex queries over the set of instances maintained in secondary storage. *DL-Lite* the usual DL logical tasks are polynomial in the amount of the TBox, and query answering is polynomial in the amount of the ABox (i.e., in data difficulty). To the best of knowledge, this is the first result of polynomial data difficulty for query answering over DL knowledge bases. A distinguished visage of logic is to allow for a partitions between TBox and ABox logic during query evaluation: the part of the process requiring TBox logic is self-determining of the ABox, and the some part of the process requiring access to the ABox which can be carried out by an SQL engine, thus taking benefit of the query optimization strategies provided by current DBMSs.

Keywords-component:*knowledg_ebase,Description_Logic,binary_relation*

*****

I. INTRODUCTION

Description Logics (DLs) is apprehensive with the trade-off between expressive power and computational difficulty of sound and complete reasoning. Survey carried out in the past on this topic has shown that many DLs with capable, i.e., worst case of polynomial time, logical algorithms lack modeling power required in capturing theoretical models and basic ontology languages, while most DLs with sufficient modeling power undergo from inherently worst case exponential time manners of reasoning [4, 5]. Even if the requirement of polynomially tractable reasoning might be less strict when dealing with comparatively small ontologies, we consider that the need of efficient reasoning algorithms is of principal significance when the ontology system is used to manage large amount of objects (e.g., from thousands to millions of instances). Several important applications uses ontologies now a days. For example, in the Semantic Web, ontologies are often used to explain the important concepts of Web repositories, and such repositories may include very large data sets. In such cases, two necessities come out that are typically overlooked in DLs. First, the number of instance in the knowledge bases requires organization instances of concepts (i.e., ABoxes) in secondary storage. Second, major 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. Regrettably, whenever the difficulty of logic is exponential in the size of the instances [11], there is little hope for effective instance management and query answering algorithms. In this paper they propose a new DL, called *DL-Lite*, specially modified to capture basic ontology languages, while keeping low difficulty of logic, in particular, polynomial in the size of the instances in the knowledge base. Logic here means computing subsumption between concepts, and checking satisfiability of the whole knowledge base, answering those queries which are difficult over the set of instances maintained in secondary storage. Some of the contribution are :

1. They define *DL-Lite*, and show that it is prosperous sufficient to capture a major ontology language. Although at a first sight *DL-Lite* appears like very simple DL, the kind of modeling constructs in their logic makes it suitable for expressing a diversity of representation language broadly adopted in different contexts, such as basic ontology languages, abstract data models (e.g., Entity-Relationship [2]), and object-oriented formalisms (e.g.,
basic UML class diagrams[3]).

2. For such a DL they gave narrative logic techniques for a multiplicity of tasks, including conjunctive query answering and control between conjunctive queries over concepts and roles. Their management is focused especially on the problem of answering conjunctive queries over a knowledge base. This is one of the few results on answering difficult queries over a DL knowledge base [11]. In fact, answering conjunctive queries over a knowledge base is a difficult problem, even in the case of *DL-Lite*, where the mixture 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 "variablefree" fragments of first-order logic represented by DLs.

3. An one main feature of this approach is that it is completely suitable to representing ABox assertions managed in secondary storage by a Data Base Management System (DBMS). Indeed, query answering algorithm is based on the idea of growing the original query into a set of queries that can be straightly evaluated by an SQL engine over the ABox, thus taking advantage of well familiar query optimization strategies[6].

4. Analyze the difficulty of reasoning in *DL-Lite*. It show that the usual logic jobs considered in DLs can be done in polynomial time. As for query answering, computing the answers to a conjunctive query is having 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 difficulty [17].

Therefore, the difficulty of answering queries is no worse than established query evaluation in relational databases.
An important feature of this approach is that it is completely suitable to representing ABox assertions managed in secondary storage by a Data Base Management System (DBMS).

II. DL-LITE

As usual in DLs, *DL-Lite* allows for denoting binary relations between objects. *DL-Lite* concepts are defined as follows:

B ::= A | ∃R | $\exists R^{-}$
C ::= B | ¬B | C1 ∏ C2

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 ∃R, specifically, 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 (common) concept. Note that thwy uses reversal of basic concepts only, and we do not allow for disjunction.

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

B ∈ C *inclusion assertions*
(funct R), (funct R^{-}) *functionality assertions*

An inclusion declaration expresses that a basic concept is subsumed by a common concept, while a functionality assertion expresses the (inclusive) 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*) *membership assertions*

where a and b are constants. These assertions utter 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 fairly simple from the language point of vision, it allows for querying the extensional knowledge of a knowledge of a KB in a much extra authoritative way than common DLs, in which only membership to a concept or to a role can be asked. *E*xpressly, *DL-Lite* allows for using conjunctive queries of random complexity. A conjunctive query (CQ) *q* over a knowledge base *K* is an look of the form:

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

where $\vec{x}$ are the so-called *well-known variables*, $\vec{y}$ are existentially quantified variables called the *non- well-known* 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 *K*, and *z*, *z*1, *z*2 are constants in *K* or variables in $\vec{x}$ or $\vec{y}$. Sometimes, for simplifying details, we will use the Datalog sentence structure, and write queries of the above form as *q*($\vec{x}$) ←*body*($\vec{x}$, $\vec{y}$) where the existential quantification ∃ $\vec{y}$ has been ready inherent, and the symbol "," is used for conjunction in *body*($\vec{x}$, $\vec{y}$).

The semantics of *DL-Lite* is specified in terms of interpretations over a permanent endless *domain* Δ. They assume to have one constant for each object, denoting accurately that object. In other terms, they have *standard names* [15], and they will not differentiate among the alphabet of constants and Δ.

An *interpretation* $I = (\Delta,.^{I})$ consists of a initial order arrangement over Δ with an *interpretation function* $.^{I}$ such that:

$A^{I} \sqsubseteq \Delta$ $R^{I} \sqsubseteq \Delta \times \Delta$
$(\neg B)^{I} = \Delta \setminus B^{I}$ $(\exists R)^{I} = \{c | \exists c'. (c, c') \in R^{I}\}$
$(C_1 \sqcap C_2)^{I} = C_1^{I} \cap C_2^{I}$ $(\exists R^{-})^{I} = \{c | \exists c'. (c, c') \in R^{I}\}$

An interpretation *I* is a *model* of an inclusion assertion $B \sqsubseteq C$ if and only if $B^{I} \sqsubseteq C^{I}$; *I* is a model of a functionality assertion (funct *R*) if $(c, c') \in R^{I} \wedge (c, c'') \in R \supset c' = c''$, similarly for (funct R^{-}); *I* is a form of a membership assertion *B*(*a*) (resp. *R*(a, *b*)) if $a \in BI$ (resp. (a, *b*) $\in R^{I}$). A *model of a KB K* is an interpretation *I* that is a model of all the assertions in *K*. A KB is *satisfiable* if it has at least one model. A KB *K sensibly implies* an assertion α if all the models of *K* are also models of . A query *q*($\vec{x}$)∃ ←$\vec{y}$,*conj* ($\vec{x}$, $\vec{y}$) is interpreted in an interpretation *I* as the set q^{I} of tuples ~$c \in \Delta \times \ldots \times \Delta$ such that when replace with the variables $\vec{x}$ with the constants ~*c*, the method ∃ $\vec{y}$.*conj* ($\vec{x}$, $\vec{y}$) evaluates to true in *I*.

Ever since *DL-Lite* deals with conjunctive queries, the vital logic services that are of interest are:

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

Example 1 Let the infinitesimal concepts *Professor* and *Student*, the roles *TeachesTo* and *HasTutor*, and the following *DL-Lite* TBox *T* :

Professor $\sqsubseteq \exists$*TeachesTo* *Student* $\sqsubseteq \exists$*HasTutor*
$\exists$*TeachesTo*$^- \sqsubseteq$ *Student* $\exists$*HasTutor*$^- \sqsubseteq$ *Professor*
Professor $\sqsubseteq \neg$*Student* (funct *HasTutor*).

Suppose that the ABox A contains just the assertion (John,Mary). At last, think the query $q(x) \leftarrow TeachesTo(x,y)$, $HasTutor(y, z)$, asking for professors that teach to students that have a tutor.

Even though prepared with higher logic services, at initial sight *DL-Lite* might seem rather feeble in modeling intensional knowledge, and therefore of partial use in practice. Although the ease of its language and the specific form of inclusion assertions acceptable, *DL-Lite* is capable to capture the major notions (though not all, obviously) of both ontologies, and of intangible modeling formalisms used in databases and software engineering. In particular, *DL-Lite* assertions allow to specify *ISA*, e.g., stating that concept $A1$ is subsumed by concept $A2$, using $A1 \sqsubseteq A2$; *disjointness*, e.g., between concepts $A1$ and $A2$, using $A1 \sqsubseteq \neg A2$; *role-typing*, e.g., stating that the first (resp., second) component of the relation R is an instance of $A1$ (resp., $A2$), using $\exists R \sqsubseteq A1$ (resp., $\exists R^- \sqsubseteq A2$); *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 (funct R) and (funct R^-). Notice that DL-Lite is a firm subset of OWL Lite, the fewer expressive sublanguage of OWL, which presents various constructs (e.g., some kinds of role limits) that are non expressible in DL-Lite, and that make logic in OWL Lite non- well-mannered in general.

III. REASONING IN DL-LITE

It can be revealed that *query containment* can be reformulated as *query answering* using techniques similar to the ones in [1].

First tackle some introduction issues, and then describe the query reformulation algorithm PerfectRef, which is at the heart of query evaluation algorithm Answer. Finally, address accuracy and difficulty issues.

KB normalization It indicate by Normalize(K) the *DL-Lite* KB obtained by transforming the KB $K = (T, A)$ as follows. The ABox A is stretched out by adding to A the assertions $\exists R(a)$ and $\exists R^-(b)$ for each $R(a, b) \in A$.

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

The TBox T resulting from such a transformation contains assertions of the form (*i*) $B_1 \sqsubseteq B_2$, where B_1 and B_2 are essential concepts (i.e., each of them is either an atomic or an existential concept), which call *positive inclusions (PIs)*; (*ii*) $B_1 \sqsubseteq \neg B_2$, where $B1$ and $B2$ are basic concepts, which call *negative inclusions (NIs)*; (*iii*) functionality assertions on roles of the form (funct R) or (funct R^-).

Then, the TBox T is expanded by computing all (nontrivial) NIs between essential concepts indirect by T . More precisely, the TBox T is closed with respect to the following assumption rule: if $B_1 \sqsubseteq B_2$ occurs in T and either $B_2 \sqsubseteq \neg B_3$ or $B_3 \sqsubseteq \neg B_2$ occurs in T (where $B1$,$B2$,$B3$ are arbitrary basic concepts), then add $B1 \sqsubseteq \neg B3$ to T . It can be shown that, after the over closure of T , for each couple of essential concepts $B1$, $B2$, they have that $T \models B_1 \sqsubseteq \neg B_2$ iff either $B1 \sqsubseteq \neg B_2 \in T$ or $B_2 \sqsubseteq \neg B_1 \in T$.

It is immediate to confirm that, for each *DL-Lite* KB K, Normalize(K) is equal to K, in the intellect that the set of models of K coincides with that of Normalize(K). In the following, without failure of generality we assume that every concept name or role name occurring in A also occurs in T .

ABox storage Formerly the ABox is normalized, amass it underneath the control of a DBMS, in order to successfully handle objects in the knowledge base by means of an SQL engine. To this aim, they construct a relational database which faithfully represents a normalized ABox A. More precisely,

- for each essential concept B occurring in A, define a relational table tab_B of arity 1, such that $\langle a \rangle \in tab_B$ if and only if $B(a) \in A$;
- for every role R occurring in$A$, define a relational table tab_R of arity 2, such that $\langle a, b \rangle \in tab_R$ if and only if $R(a, b) \in A$. They denote with DB(A) the relational database thus constructed.

KB satisfiability The algorithm Consistent takes as input a normalized KB $K = (T , A)$ and verifies the following situation:

(i) there exists a NI $B_1 \sqsubseteq \neg B_2$in T and a steady a such that the assertions $B_1(a)$ and $B_2(a)$ fit in to A;
(ii) there exists an assertion (funct R) (respectively, (funct R^-)) in 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 A.

Casually, situation (i) corresponds to examination whether A openly contradicts some NI in T , and situation (ii) corresponds to check whether A violates some functionality assertion in T . If one of the above conditions holds, then the

algorithm returns *false* (i.e., K is not satisfiable); or else, the algorithm returns *true*.

Notably, the algorithm verifies such situation by affectation to DB(A) appropriate conjunctive queries expressed in SQL. For example, situation (i) holds for a given NI $B_1 \sqsubseteq \neg B_2$ if and only if the query $q(x) \leftarrow tab_{B1}(x), tab_{B2}(x)$ has a non-empty if and only if the query $q(x) \leftarrow tab_R(x; y), tab_R(x, z), y \neq z$ has a non-empty answer in DB(A), where $\neq$ is the "not equal" predicate of SQL. Notice that the algorithm does not think about the PIs occurring in T through its execution. Indeed, they will show that PIs do not concern the stability of a *DL-Lite* KB, if the TBox is normalized.

Query reformulation Query reformulation is at the spirit of query answering technique. Given the limited significant power of *DL-Lite* TBoxes, it may seem that in arrange to answer a query q over a KB K, we could simply construct a limited first-order structure on the origin of K, and then estimate the query as an expression over this first-order structure. Actually, it is feasible to show that this is not the case. In exacting, it can be exposed that, in common, given a KB K, there exists no finite structure S such that, for all conjunctive query q, the set of answers to q over K is the answer of evaluating q over S. This property demonstrates that answering queries in *DL-Lite* goes beyond both propositional logic and relational databases. The essential suggestion of technique is to reformulate the query captivating into version the TBox: in exacting, given a query q over 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 K, as if the ABox were a easy relational database. Since the size of q' does not depend on the ABox, the data difficulty of the whole query answering algorithm is polynomial.

In case of an atom in a query is *bound* if it corresponds to either a distinguished variable or a shared variable, i.e., a variable taking place at least two times in the query body, or a constant, while that it is *unbound* if it corresponds to a non-distinguished non-shared variable (as usual, the symbol _ used to represent non-distinguished non-shared variables). Notice that, an atom of the type $\exists R(x)$ (resp. $\exists R^{-}(x)$) has the similar meaning as $R(x,\_)$ (resp. $R(\_, x)$).

API I *is applicable to an atom* $B(x)$, if I has B in its right-hand side, and I *is applicable to an atom* $R(x1, x2)$, if either (i) $x2 = \_$ and the right-hand side of I is $\exists R$, or (ii) $x1 = \_$ and the right-hand side of I is $\exists R^{-}$. Approximately 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 T are unary, they are by no means applicable to atoms with two bound arguments. It 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.

Query evaluation In sort to calculate the answers to q over the KB $K = (T, A)$, it require to evaluate the set of conjunctive queries P formed by the algorithm PerfectRef over the ABox A. Obviously, in doing so, to develop the relational database DB(A). For this aim, they require to convert every query q in P into an SQL query expressed over DB(A). The conversion is theoretically very simple. The only non-trivial case concerns binary atoms with limitless terms: for an atom of the form $R(\_, x)$, uses 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 $R(x, \_)$). Each and every one SQL queries obtained from P, together with the views introduced in the conversion, denoted by SQL(P), can be easily dispatched to an SQL query engine and evaluated over DB(A).

Example 1 (contd.). Since ABox A contains only the declaration *HasTutor*(John;Mary), it is minor to begin satisfiability of K . Then, by executing Answer($q;K$), first get Normalize(K), which is computed by adding together to T all NIs implied by T , i.e.,:

$$\exists TeachesTo^{-} \sqsubseteq \neg Professor \qquad \exists HasTutor^{-} \sqsubseteq \neg Student.$$

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

IV. DISCUSSION AND RELATED WORK

DL-Lite is a portion of expressive DLs with assertions and inverses studied in the 90's (see [4] for an overview), which are at the foundation of present ontology languages such as OWL, and for which optimized automated logic systems such as Fact and Racer have been developed. Definitely, one could use, off-the-shelf, a system like Racer to execute KB satisfiability, instance checking (of concepts), and logical inference of inclusion assertions in *DL-Lite*. Also, reasoning with conjunctive queries in these DLs has been studied (see e.g. [11]), though not yet implemented in systems. Regrettably, the reasoning events for these DLs are all EXPTIME-hard, and further significantly they are not tailored AAAI-05 / 606 towards obtaining firm difficulty bounds with respect to data difficulty. Conjunctive queries joint with DLs were also considered in [16, 13], but again data complication was not the major worry. There has been a bundle of work in DLs on the border line between polynomial and exponential reasoning. This work first determined on DLs without the TBox factor of the KB, and led to the growth of simple DLs, such as *ALN*, that declare polynomial instance checking. However, for negligible variants of *ALN*, such as *ALE* , $FLE_{¡}$, and *ALU*, instance checking, and therefore conjunctive query answering, is coNP-complete in data complication [12].

Then permit for repeated inclusion assertions in the KB, then even subsumption in CLASSIC and *ALN* becomes inflexible [9]. Examine that *DL-Lite* does permit for repeated assertions without declining into intractability. In reality, they can impose the repeated propagation of the existence of an R-successor using the two *DL-Lite* inclusion assertions $A \sqsubseteq \exists R$, $\exists R^{-} \sqsubseteq A$. The restriction forced on a model is like to the one forced by the *ALN* repeated 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 limitations on the use of the $\forall R.C$ construct, which, if used jointly with inclusion assertions, immediately would lead to intractability [9]. The 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 permit for expressing unique forms of inclusion dependencies, numerous

keys on relations, and exclusion dependencies. The results that report here show that *DL-Lite* inclusion assertions form one of the largest class of ICs for which query answering remains polynomial.

CONCLUSIONS

DL-Lite, a latest DL purposely modified to imprison theoretical data models and essential ontology language while keeping the worst-case difficulty of sound and whole logic obedient. It paying attention on binary roles only, other than it is feasible to widen logic techniques to *n*-ary relations without loosing their pleasant computational properties. Working on further motivating extensions to *DL-Lite*, such as the preface of subset constraints on roles. The results of [10] entail that verdict an adjustment of query answering technique is going to be a unbreakable trouble.

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. Cal`ı, 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. *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.