Answering Conjunctive Queries Over DL Aboxes: a Preliminary Report

Ian Horrocks and Sergio Tessaris
Department of Computer Science
University of Manchester, UK
{horrocks|tessaris}@cs.man.ac.uk

1 Introduction

A serious shortcoming of many Description Logic based knowledge representation systems is the inadequacy of their query languages. In this paper we present a novel technique that can be used to provide an expressive query language for such systems. A typical description logic (DL) knowledge base (KB) is made up of two parts, a terminological part (the Tbox) and an assertional part (the Abox), each part consisting of a set of axioms. The Tbox asserts facts about *concepts* (sets of objects) and *roles* (binary relations), usually in the form of inclusion axioms, while the Abox asserts facts about *individuals* (single objects), usually in the form of instantiation axioms.

Recent years have seen significant advances in the design of sound and complete reasoning algorithms for DLs with both expressive logical languages and unrestricted Tboxes, i.e., those allowing arbitrary concept inclusion axioms [1, 11, 7]. Moreover, systems using highly optimised implementations of (some of) these algorithms have also been developed, and have been show to work well in realistic applications [10, 16]. While most of these have been restricted to terminological reasoning (i.e., the Abox is assumed to be empty), attention is now turning to the development of both algorithms and (optimised) implementations that also support Abox reasoning [9, 20].

Although these systems provide sound and complete Abox reasoning for very expressive logics, their utility is limited w.r.t. earlier DL systems by their very weak Abox query languages. Typically, these only support instantiation (is an individual i an instance of a concept C), realisation (what are the most specific concepts i is an instance of) and retrieval (which individuals are instances of C). This is in contrast to a system such as Loom where a full first order query language is provided, although based on incomplete reasoning algorithms [15].

The reason for this weakness is that, in these expressive logics, all reasoning tasks are reduced to that of determining KB satisfiability (consistency). In particular, instantiation is reduced to KB (un)satisfiability by transforming the query into a negated assertion; however, this technique cannot be used (directly) for queries involving roles because these logics do not support role negation.

In this paper we present a technique for answering such queries using a more sophisticated reduction to KB satisfiability. We then show how this technique can be extended to determine if an arbitrary tuple of individuals (i.e., not just a singleton or pair) satisfies a disjunction of conjunctions of concept and role membership assertions that can contain both constants (i.e., individual names) and variables. This provides a powerful query language, similar to the conjunctive queries typically supported by relational databases,[1] that allows complex Abox structures (e.g., cyclical structures) to be retrieved by using variables to enforce co-reference. For example, the query

$$\langle x, y \rangle \quad \leftarrow \quad \langle z, \mathsf{Bill} \rangle{:}\mathsf{Parent} \wedge \langle z, x \rangle{:}\mathsf{Parent} \wedge \langle z, y \rangle{:}\mathsf{Parent} \wedge \langle x, y \rangle{:}\mathsf{Hates} \qquad (1)$$

would retrieve all the pairs of hostile siblings in Bill's family.[2]

In this paper we focus on answering boolean queries, i.e., determining if a query is true with respect to a KB. Retrieval can be easily (although inefficiently) turned into a set of boolean queries for all candidate tuples. Note that in available systems, the retrieval problem is similarly reduced to instantiation.[3] It is important to stress the fact that, given the expressivity of DLs, query answering cannot simply be reduced to model checking as in the database framework. This is because KBs may contain nondeterminism and/or incompleteness, making it infeasible to use an approach based on minimal models. In fact, query answering in the DL setting requires the same reasoning machinery as logical derivation.

An important advantage with the technique presented here is that it is quite generic, and can be used with any DL providing general inclusion axioms where instantiation can be reduced to KB satisfiability. It could therefore be used to significantly increase the utility of Abox reasoning in a wide range of existing (and future) DL implementations.

2 Preliminaries

Although the query answering technique is quite general, it will simplify the presentation if we consider a concrete DL language. We will use the language $\mathcal{ALC}$ [19] as it is widely known, is sufficiently expressive for our purposes (in particular, it is closed under negation) and is a subset of the logics implemented in most "state of the art" DL systems, i.e., those based on highly optimised tableaux algorithms [10, 16, 9].

$\mathcal{ALC}$ concepts are built using a set of concept names (NC) and role names (NR). If $A \in \mathsf{NC}$ is a concept name, $R \in \mathsf{NR}$ is a role name, and C_1, C_2 are concepts then

[1]It is inspired by the use of Abox reasoning to decide conjunctive query containment [12, 5].

[2]Note that a sound and complete KB satisfiability algorithm will guarantee sound and complete query answers.

[3]Apart from not very expressive DL languages (see for example [17]).

the expressions $A, \top, \bot, \neg A \mid C_1 \sqcap C_2, C_1 \sqcup C_2, \forall R.C, \exists R.C$ are concept as well. The meaning of concepts is given by a Tarski style model theoretic semantics using *interpretations*. An interpretation $\mathcal{I}$ is a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is the domain and $\cdot^{\mathcal{I}}$ an interpretation function. The function $\cdot^{\mathcal{I}}$ maps each concept name in NC to a subset of $\Delta^{\mathcal{I}}$ and each role name in NR to a binary relation over $\Delta^{\mathcal{I}}$ (a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$) such that the following equations are satisfied:

$$\begin{array}{rcll}
\top^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \quad (C_1 \sqcap C_2)^{\mathcal{I}} & = \; {C^{\mathcal{I}}}_1 \cap {C^{\mathcal{I}}}_2 \\
\bot^{\mathcal{I}} & = & \emptyset \quad\;\; (C_1 \sqcup C_2)^{\mathcal{I}} & = \; {C^{\mathcal{I}}}_1 \cup {C^{\mathcal{I}}}_2 \\
 & & (\neg A)^{\mathcal{I}} & = \; \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} & = & \multicolumn{2}{l}{\left\{ i \in \Delta^{\mathcal{I}} \mid \forall j.\, (i,j) \in R^{\mathcal{I}} \Rightarrow j \in C^{\mathcal{I}} \right\}} \\
(\exists R.C)^{\mathcal{I}} & = & \multicolumn{2}{l}{\left\{ i \in \Delta^{\mathcal{I}} \mid \exists j.\, (i,j) \in R^{\mathcal{I}} \wedge j \in C^{\mathcal{I}} \right\}}
\end{array}$$

2.1 DL knowledge bases

A DL knowledge base is a pair $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is called the *Tbox* and $\mathcal{A}$ is called the *Abox*.

The Tbox, or terminology, is a set of assertions about concepts of the form $C \sqsubseteq D$, where C and D are concepts.[4] An interpretation $\mathcal{I}$ *satisfies* $C \sqsubseteq D$ (written $\mathcal{I} \models C \sqsubseteq D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ and it satisfies a Tbox $\mathcal{T}$ (written $\mathcal{I} \models \mathcal{T}$) if it satisfies every assertion in $\mathcal{T}$.

The Abox, or assertional part, is a set of assertions about a set of individuals names NI. These assertions are of the form $a{:}C$ and $\langle a, b \rangle{:}R$, where a, b are names in NI, C is a concept and R is a role. The semantics of the Abox is given by extending the interpretation function $\cdot^{\mathcal{I}}$ to map each individual name in NI to a single element of $\Delta^{\mathcal{I}}$. An interpretation $\mathcal{I}$ satisfies $a{:}C$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$, it satisfies $\langle a, b \rangle{:}R$ iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ and it satisfies an Abox $\mathcal{A}$ (written $\mathcal{I} \models \mathcal{A}$) if it satisfies every assertion in $\mathcal{A}$.

An interpretation satisfies a knowledge base $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ (written $\mathcal{I} \models \Sigma$) if it satisfies both $\mathcal{T}$ and $\mathcal{A}$; a knowledge base is said to be satisfiable iff there exists at least one non-empty interpretation satisfying it. Using the definition of satisfiability, an assertion X is said to be a *logical consequence* of a KB Σ (written $\Sigma \models X$) iff X is satisfied by every interpretation that satisfies Σ.

The semantics of DL Aboxes often includes a so called *unique name assumption*: an assumption that the interpretation function maps different individual names to different elements of the domain (i.e., $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ for all $a, b \in \mathsf{NI}$ such that $a \neq b$). Our approach does not rely on such an assumption, and can be applied to DLs both with and without the unique name assumption.

[4] $C \doteq D$ is sometimes used as an abbreviation for the pair of assertions $C \sqsubseteq D$ and $D \sqsubseteq C$.

2.2 Queries

In this paper we will focus on conjunctive queries, but the extension to disjunctions of conjunctive queries is relatively straightforward [13]. A key feature of conjunctive queries is that they may contain variables, and we will assume the existence of a set of variables V that is disjoint from the set of individual names, i.e., $\mathsf{V} \cap \mathsf{NI} = \emptyset$. We distinguish *boolean* conjunctive queries of the form $q_1 \wedge \ldots \wedge q_n$, where $q_1, \ldots, q_n$ are query terms. Each query term q_i is of the form $x{:}C$ or $\langle x, y\rangle{:}R$, where C is a concept, R is a role and x, y are either individual names or variables. Given a KB Σ, an interpretation $\mathcal{I}$ of Σ satisfies a query $\mathcal{Q}$ iff the interpretation function can be extended to the variables in $\mathcal{Q}$ in such a way that $\mathcal{I}$ satisfies every term in $\mathcal{Q}$. A query $\mathcal{Q}$ is *true* w.r.t. Σ (written $\Sigma \models \mathcal{Q}$) iff every interpretation that satisfies Σ also satisfies $\mathcal{Q}$. For example, the query

$$\langle \mathsf{Bill}, y\rangle{:}\mathsf{Parent} \wedge \langle y, z\rangle{:}\mathsf{Parent} \wedge z{:}\mathsf{Male} \tag{2}$$

is true w.r.t. a KB Σ iff it can be inferred from Σ that Bill has a grandson. Note that query truth value and the idea of logical consequence are strictly related. In fact, a boolean query is true w.r.t. a KB iff it is logical consequence of the KB.

In the following, we will only consider how to answer boolean queries. Retrieving sets of tuples can be achieved by repeated application of boolean queries with different tuples of individual names substituted for variables. For example, the answer to the retrieval query $\langle x, y, z\rangle \leftarrow \mathcal{Q}$ w.r.t. a KB Σ is the set of tuples $\langle a, b, c\rangle$, where a, b, c are individual names occurring in Σ, such that $\Sigma \models \mathcal{Q}'$ for the boolean query $\mathcal{Q}'$ obtained by substituting a, b, c for x, y, z in $\mathcal{Q}$. Of course the naive evaluation of such a retrieval could be prohibitively expensive, but would clearly be amenable to optimisation. For example, let us consider the query (1). In many DLs the expressivity of the Abox for roles is very limited: in $\mathcal{ALC}$, for example, a KB implies a query term like $\langle z, \mathsf{Bill}\rangle{:}\mathsf{Parent}$, where the second argument is an individual name, only if there is an explicit assertion of this form in the Abox. Therefore we may use role assertions in the Abox to reduce the number of candidates among the individual names.

We will show how to answer boolean queries in two steps. Firstly, we will consider conjunctions of terms containing only individual names appearing in the KB; secondly, we will show how this basic technique can be extended to deal with variables.

3 Queries with multiple terms

In this section we will consider queries expressed as a conjunction of concept and role terms built using only names appearing in the KB (i.e. without variables).

As we have already seen, logical consequence can easily be reduced to a KB satisfiability problem if the query contains only a single concept term (this is the standard instantiation problem). For example, $\mathsf{Tom{:}Person}$ is a logical consequence of the KB $\langle\{\mathsf{Student} \sqsubseteq \mathsf{Person}\}, \{\mathsf{Tom{:}Student}\}\rangle$ iff the KB

$$\langle\{\mathsf{Student} \sqsubseteq \mathsf{Person}\}, \{\mathsf{Tom{:}Student}, \mathsf{Tom{:}}\neg\mathsf{Person}\}\rangle$$

is not satisfiable. This can be generalised to queries containing conjunctions of concept terms simply by transforming the query test into a set of (un)satisfiability problems: a conjunction $a_1{:}C_1 \wedge \ldots \wedge a_n{:}C_n$ is a logical consequence of a KB iff each $a_i{:}C_i$ is a logical consequence of the KB.

However, this simple approach cannot be used in our case since a query may also contain role terms. Instead, we will show how simple transformations can be used to convert every role term into a concept term. We call this procedure *rolling up* a query.

The rationale behind rolling up can easily be understood by imagining the availability of the DL `one-of` operator, which allows the construction of a concept containing only a single named individual [18]. The standard notation for such a concept is $\{a\}$, where a is an individual name, and the semantics is given by the equation $\{a\}^{\mathcal{I}} = \left\{a^{\mathcal{I}}\right\}$. For example, the expression $\{\mathsf{Bill}\}$ represents a concept containing only the individual Bill (i.e., $\{\mathsf{Bill}\}^{\mathcal{I}} = \left\{\mathsf{Bill}^{\mathcal{I}}\right\}$).

Using the `one-of` operator, the role term $\langle\mathsf{John}, \mathsf{Bill}\rangle{:}\mathsf{Brother}$ can be transformed in the equivalent concept term $\mathsf{John}{:}(\exists\mathsf{Brother}.\{\mathsf{Bill}\})$. Furthermore, other concept terms asserting additional facts about the individual being rolled up (Bill in this case) can be absorbed into the rolled up concept term. For example, the conjunction

$$\langle\mathsf{John}, \mathsf{Sally}\rangle{:}\mathsf{Parent} \wedge \mathsf{Sally}{:}\mathsf{Female} \wedge \mathsf{Sally}{:}\mathsf{PhD}$$

can be transformed into $\mathsf{John}{:}\exists\mathsf{Parent}.(\{\mathsf{Sally}\} \sqcap \mathsf{Female} \sqcap \mathsf{PhD})$. The absorption transformation is not strictly necessary for queries without variables, but it serves to reduce the number of satisfiability tests needed to answer the query (by reducing the number of conjuncts), and it will be required with queries containing variables. By applying rolling up to each role term, an arbitrary query can be reduced to an equivalent one which contains only concept terms, and which can be answered using a set of satisfiability tests as described above.

However, the logic we are using does not include the `one-of` operator, nor is it provided by any state of the art DL system (in fact the decidability of expressive DLs including this operator is still an open problem). Fortunately, we do not need the full expressivity of `one-of`, and in our case it can be "simulated". The technique used is to substitute each occurrence of `one-of` with a new concept name not appearing in the knowledge base. These new concept names must be different for each individual in the query, and are called the *representative* concepts of the individuals (written P_a, where a is the individual name). In addition, assertions which ensure that each individual is an instance of its representative concept must be added to the knowledge base (e.g., $\mathsf{Bill}{:}P_{\mathsf{Bill}}$). In general, a representative concept cannot be used in place of `one-of` because it can have instances other than the individual which it represents (i.e., ${P_a}^{\mathcal{I}} \supseteq \left\{a^{\mathcal{I}}\right\}$). However, representative concepts can be used instead of `one-of` in our reduced setting. In particular, the conjunction $\langle a,b\rangle{:}R \wedge b{:}C$ is a logical consequence of a given knowledge base if and only if $a{:}\exists R.(P_b \sqcap C)$ is a logical consequence of the very same knowledge base augmented by the assertion $b{:}P_b$.

Due to space considerations, we will not reproduce here a formal proof of this theorem, or of any of the other transformations used in this paper: full details can be found in [12].

4 Queries with variables

In this section we show how variables can be introduced in this framework by using a more complex rolling up procedure in order to obtain a similar reduction to the KB (un)satisfiability problem. Variables can be used exactly as individual names, but their meaning is as "place-holders" for unknown elements of the domain. Because variables may be interpreted as any element of the domain, they cannot simply be considered as individual names to which the unique name assumption does not apply; nor can they be treated as referring only to named individuals, giving the possibility of nondeterministically substituting them with names in the KB. In fact the query (2) is true w.r.t. both the KBs

$$\langle\emptyset,\left\{\begin{array}{l}\langle\mathsf{Bill},\mathsf{Mary}\rangle{:}\mathsf{Parent},\\ \langle\mathsf{Mary},\mathsf{Tom}\rangle{:}\mathsf{Parent},\\ \mathsf{Tom}{:}\mathsf{Male}\end{array}\right\}\rangle \quad\text{and}\quad \langle\emptyset,\{\mathsf{Bill}{:}\exists\mathsf{Parent}.(\exists\mathsf{Parent}.\mathsf{Male})\}\rangle,$$

but for the first KB the variables can be substituted by the individual names Mary and Tom, while in the second case the variables may need to be interpreted as elements of the domain that are not the interpretations of any named individuals.

Answering queries containing variables involves a more sophisticated rolling up technique. For example, let us consider the terms $\langle y,z\rangle{:}\mathsf{Parent}$ and $z{:}\mathsf{Male}$ of query (2). If z were an individual name, then the terms could be rolled up as $y{:}\exists\mathsf{Parent}.(P_z \sqcap \mathsf{Male})$, but this is not an equivalent query when z is a variable name because z can be interpreted as any element of the domain, not just an element of ${P_z}^{\mathcal{I}}$. However, since in this case z is no longer referred to in any other place in the query, there is no other constraint on how an interpretation can be extended w.r.t. z, so the concept $\top$ (whose interpretation is always the whole domain) can be used instead of P_z. The resulting concept term is $y{:}\exists\mathsf{Parent}.(\top \sqcap \mathsf{Male})$, which can be simplified to $y{:}\exists\mathsf{Parent}.\mathsf{Male}$. The same procedure can now be applied to y, thereby reducing query (2) to the single concept term $\mathsf{Bill}{:}\exists\mathsf{Parent}.(\exists\mathsf{Parent}.\mathsf{Male})$.

In order to show how this procedure can be more generally applied, it will be useful to consider the directed graph induced by the query, i.e., a graph in which there is a node x for each individual or variable x in the query, and an edge R from node x to node y for each role term $\langle x,y\rangle{:}R$ in the query. It is easy to see that the rolling up procedure can be used to eliminate variables from any tree-shaped part of a query by starting at the leaves and working back towards the root (this is similar to the notion of descriptive support described in [17]). The fact that rolling up should start from leaves is essential for correctness: for example, rolling up query (2) in the reverse order would lead to the non-equivalent $\mathsf{Bill}{:}\exists\mathsf{Parent}.\top \wedge y{:}\exists\mathsf{Parent}.\top \wedge z{:}\mathsf{Male}$.

However, this simple procedure cannot be applied to parts of the query that contain cycles, or where more than one edge enters a node corresponding to a variable (i.e., with terms like $\langle x, z\rangle{:}R \wedge \langle y, z\rangle{:}S$).

Let us consider the case where a variable is involved in a cycle, e.g., the simple query

$$\langle x, y\rangle{:}\mathsf{Path} \wedge \langle y, z\rangle{:}\mathsf{Path} \wedge \langle z, x\rangle{:}\mathsf{Path} \tag{3}$$

which tests the KB for the presence of a loop involving the role Path. Rolling up one of the terms does not help, because the resulting query

$$\langle x, y\rangle{:}\mathsf{Path} \wedge \langle y, z\rangle{:}\mathsf{Path} \wedge z{:}\exists\mathsf{Path}.P_x$$

still contains another reference to the variable x, and replacing P_x with $\top$ would result in a non-equivalent query that no longer contained a cycle. Moreover, it is obvious that there is no way to roll up the query in order to obtain a single occurrence of any of the three variables.

This problem can be solved by exploiting the tree model property of the logic. Given this property, we know that Tbox assertions alone cannot constrain all models to be cyclical (if there is a model, then there is a tree model), so any cycle that might satisfy a cyclical query must be explicitly asserted in the Abox. Moreover, given the restricted expressivity of role assertions (i.e., that they apply only to atomic role names), cycles enforced in every interpretation must be composed only of elements interpreting individual names occurring in the Abox. Therefore, before applying the rolling up procedure, a variable occurring in a cycle can be nondeterministically substituted with an individual name occurring in the Abox.

The intuition behind this property can be understood by considering that, given an arbitrary interpretation satisfying the cycle only with elements not corresponding to individual names, a new interpretation can be build where the cycle is split by duplicating one or more of the involved elements. This new interpretation can be defined in such a way that it still satisfies the KB, but no longer contains the required cycle. This duplication can be performed only if the elements are not "fixed" individual names and assertions in the Abox. A similar argument can be used w.r.t. variables appearing as the second argument of more than one role term, e.g., the variable z in the query $\langle x, z\rangle{:}R \wedge \langle y, z\rangle{:}S$. Such variables can also be dealt with by nondeterministically substituting them with individual names occurring in the Abox.

We have seen how role terms containing variables can be rolled up into concept terms, but these may still be of the form $x{:}C$, where x is a variable. For example, the query $\langle x, y\rangle{:}\mathsf{Parent}$, where x and y are variables, can only be reduced to the single term x:$\exists$Parent.$\top$. In this case we need to verify that the interpretation of the concept $\exists$Parent.$\top$ is nonempty in every interpretation that satisfies the KB. In general, the interpretation of a concept C is nonempty in every interpretation that satisfies the KB $\langle \mathcal{T}, \mathcal{A} \rangle$ iff $\langle \mathcal{T} \cup \{\top \sqsubseteq \neg C\}, \mathcal{A} \rangle$ is not satisfiable.[5]

Summarising, the procedure for answering an arbitrary boolean conjunctive query is divided into two phases. Firstly, the role terms are eliminated by repeatedly applying the following rules: (i) if the graph induced by the query contains a leaf node y, then the role term $\langle x, y \rangle$:R is rolled up, and the edge $\langle x, y \rangle$ is removed from the graph; (ii) otherwise, if the graph contains a node y with multiple incoming edges, then all role terms $\langle x, y \rangle$:R are rolled up,[6] and the corresponding edges are removed from the graph; (iii) if the graph still contains edges but no leaf nodes and no confluent nodes, then it must contain a cycle. In this case a node y in a cycle is chosen (preferably an individual as this reduces nondeterminism) and rolled up as in case (ii) above. Secondly, the query (which now contains only concept terms) evaluates to true iff there is at least one nondeterministic replacement of variables with individual names such that every term is a logical consequence of the KB.

5 Discussion

In this paper we have presented a general technique for providing an expressive query language for a DL based knowledge representation system. For the sake of simplicity, we have only considered conjunctive queries over $\mathcal{ALC}$ KBs. However, the technique is general enough to be used with other DL languages, and it can be extended to deal with a disjunction of conjunctive queries [13]. Our work is motivated by the fact that many DL systems (including state of the art systems) provide no proper query language, and are only able to perform simple instantiation and retrieval reasoning tasks.

The only other comparable proposals in the literature are in the direction of integrating a DL system with Datalog [14, 8, 6]. Using Datalog as a query language can provide the ability to formulate recursive queries [4], but on the other hand, the combination with expressive DLs soon leads to undecidability [14]. In addition, a special algorithm (dependent on the DL language) must be implemented in order to reason with the resulting hybrid language.

Our approach sacrifices some expressivity in the query language, but it works with very expressive DL languages and it can easily be adapted for use with any existing (and most future) DL system equipped with the KB satisfiability reasoning service. The limits of the approach lie in the required tree model property of the underlying

[5]Some earlier DL systems cannot reason with Tbox axioms of this kind [2, 3], and this might restrict the kinds of query that could be answered.

[6]If y is a variable, then it is first replaced with an individual name chosen nondeterministically from those occurring in the the KB.

DL and in the availability of general inclusion axioms.

Our plans for future work include an implementation of the technique on top of the FaCT system [10], which has recently been extended to include Abox reasoning [20], as well as the analysis of suitable optimisations for reducing the nondeterminism due to variable substitution, both in the rolling up and the retrieval procedures.

References

[1] F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proc. of IJCAI-91*, 1991.

[2] F. Baader and B. Hollunder. KRIS: Knowledge representation and inference system. *SIGART Bulletin*, 2(3):8–14, 1991.

[3] P. Bresciani, E. Franconi, and S. Tessaris. Implementing and testing expressive description logics: a preliminary report. In *Proc. of of KRUSE'95*, pages 28–39, 1995.

[4] M. Cadoli, L. Palopoli, and M. Lenzerini. Datalog and description logics: Expressive power. In *Proc. of DBPL-97*, 1997.

[5] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proc. of PODS-98*, pages 149–158, 1998.

[6] D. Calvanese, G. De Giacomo, and M. Lenzerini. Answering queries using views in description logics. In *Proc. of DL'99*, pages 9–13, 1999.

[7] G. De Giacomo and F. Massacci. Combining deduction and model checking into tableaux and algorithms for converse-pdl. *Information and Computation*, 1998. To appear.

[8] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Al-log: Integrating datalog and description logics. *Journal of Intelligent Information Systems*, 10(3):227–252, 1998.

[9] V. Haarslev and R. Möller. An empirical evaluation of optimization strategies for abox reasoning in expressive description logics. In *Proc. of DL'99*, pages 115–119, 1999.

[10] I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of KR'98*, pages 636–647, 1998.

[11] I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. *Journal of Logic and Computation*, 9(3):385–410, 1999.

[12] I. Horrocks, U. Sattler, S. Tessaris, and S. Tobies. Query containment using a DLR ABox. LTCS-Report 99-15, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 1999a.

[13] I. Horrocks and S. Tessaris. A conjunctive query language for description logic aboxes. In *Proc. of AAAI-00*, 2000. to appear.

[14] A. Y. Levy and M.-C. Rousset. The limits on combining recursive horn rules and description logics. In *Proc. of AAAI-96*, 1996.

[15] R. M. MacGregor and D. Brill. Recognition algorithms for the LOOM classifier. In *Proc. of AAAI-92*, pages 774–779. AAAI Press, 1992.

[16] P. F. Patel-Schneider. DLP system description. In *Proc. of DL'98*, pages 87–89, 1998.

[17] M.-C. Rousset. Backward reasoning in aboxes for query answering. In *Proc. of DL'99*, pages 18–22, 1999.

[18] A. Schaerf. Reasoning with individuals in concept languages. *Data and Knowledge Engineering*, 13(2):141–176, 1994.

[19] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.

[20] S. Tessaris and G. Gough. Abox reasoning with transitive roles and axioms. In *Proc. of DL'99*, 1998.