A Conjunctive Query Language for Description Logic Aboxes

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

Abstract

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. One of the main advantages of this approach is that, being based on a reduction to knowledge base satisfiability, it can easily be adapted to most existing (and future) Description Logic implementations. We believe that providing Description Logic systems with an expressive query language for interrogating the knowledge base will significantly increase their utility.

Introduction

A 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. For example, a Tbox might contain an axiom asserting that Man is subsumed by Animal, while an Abox might contain axioms asserting that John, Peter and Bill are instances of the concept Man and that the pairs $\langle$John, Peter$\rangle$ and $\langle$Peter, Bill$\rangle$ are instances of the role Brother.

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 (Baader 1991; De Giacomo & Lenzerini 1995; Horrocks & Sattler 1999; De Giacomo & Massacci 1998). 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 (Horrocks 1998; Patel-Schneider 1998). 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 (Haarslev & Möller 1999a; Tessaris & Gough 1998).

Copyright © 2000, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

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 (MacGregor 1991), where a full first order query language is provided, although based on incomplete reasoning algorithms (MacGregor & Brill 1992).

The reason for this weakness is that, in these expressive logics, all reasoning tasks are reduced to that of determining KB satisfiability (consistency). For example, it can be inferred that John is an instance of Animal if and only if the KB is not satisfiable when an axiom is added to the Abox asserting that John is not an instance of Animal (i.e., that John is an instance of the negation of Animal). Realisation and retrieval can, in turn, be achieved through repeated application of instantiation tests. However, this technique cannot be used (directly) to infer from the above axioms that the pair $\langle$John, Bill$\rangle$ is an instance of the transitive role Brother, because these logics do not support role negation, i.e., it is not possible to assert that $\langle$John, Bill$\rangle$ is an instance of the negation of Brother.

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

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

would retrieve all the pairs of hostile siblings in Bill's fam-

[1] It is inspired by the use of Abox reasoning to decide conjunctive query containment (Horrocks *et al.* 1999a; Calvanese, De Giacomo, & Lenzerini 1998).

ily.[2]

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 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.

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 $\mathcal{ALC}$ (Schmidt-Schauß & Smolka 1991) 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 (Horrocks 1998; Patel-Schneider 1998; Haarslev & Möller 1999b).

In the following sections we will introduce and provide formal definitions for the $\mathcal{ALC}$ logic, DL knowledge bases, our query language and the various reasoning tasks with respect to knowledge bases and queries.

Description Logic $\mathcal{ALC}$

$\mathcal{ALC}$ concepts are built using a set of concept names (NC) and role names (NR). Valid concepts are defined by the following syntax:

$$C \quad ::= \quad A \mid \top \mid \bot \mid \neg A \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \forall R.C \mid \exists R.C$$

where $A \in \mathsf{NC}$ is a concept name and $R \in \mathsf{NR}$ is a role name. 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:

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

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

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.[3] 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 individuals 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.

Queries

In this paper we will focus on conjunctive queries: the extension to disjunctions of conjunctive queries can easily be accomplished using a technique sketched later on. In our framework, a key feature of 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., $\mathsf{V} \cap \mathsf{NI} = \emptyset$. A *boolean* conjunctive query $\mathcal{Q}$ is 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

$$\langle \mathsf{Bill}, y \rangle{:}\mathsf{Parent} \wedge \langle y, z \rangle{:}\mathsf{Parent} \wedge z{:}\mathsf{Male} \qquad (1)$$

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.

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

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 $\mathcal{Q}$. The naive evaluation of such a retrieval could be prohibitively expensive, but would clearly be amenable to optimisation.

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.

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, e.g., Tom:Student or ⟨Tom, CS710⟩:Enrolled.

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,

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

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 (Schaerf 1994). The standard notation for such a concept is $\{a\}$, where a is an individual name, and the semantics is given by the equation $\{a\}^{\mathcal{I}} = \{a^{\mathcal{I}}\}$. For example, the expression $\{\mathsf{Bill}\}$ represents a concept containing only the individual Bill (i.e., $\{\mathsf{Bill}\}^{\mathcal{I}} = \{\mathsf{Bill}^{\mathcal{I}}\}$).

Using the `one-of` operator, the role term ⟨John, Bill⟩: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

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

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., ${P_a}^{\mathcal{I}} \supseteq \{a^{\mathcal{I}}\}$). However, representative concepts can be used instead of `one-of` in our reduced setting, as shown by the following theorem:

Theorem 1 *Let $\Sigma = \langle \mathcal{T}, \mathcal{A}\rangle$ be a DL knowledge base, a, b two individual names in $\mathcal{A}$, R a role and $C_1, \ldots, C_n$ concepts. Given a new concept name P_b not appearing in Σ:*

$$\langle \mathcal{T}, \mathcal{A}\rangle \models \langle a, b\rangle{:}R \wedge b{:}C_1 \wedge \ldots \wedge b{:}C_n$$

if and only if

$$\langle \mathcal{T}, \mathcal{A} \cup \{b{:}P_b\}\rangle \models a{:}\exists R.(P_b \sqcap C_1 \sqcap \ldots \sqcap C_n).$$

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 (Horrocks *et al.* 1999a).

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 (1) is true w.r.t. both the KBs

$$\langle \emptyset, \left\{ \begin{array}{l} \langle \mathsf{Bill}, \mathsf{Mary}\rangle{:}\mathsf{Parent}, \langle \mathsf{Mary}, \mathsf{Tom}\rangle{:}\mathsf{Parent}, \\ \mathsf{Tom{:}Male} \end{array} \right\}\rangle$$

and

$$\langle \emptyset, \{\mathsf{Bill}{:}\exists \mathsf{Parent}.(\exists \mathsf{Parent}.\mathsf{Male})\}\rangle,$$

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 last two terms of query (1), $\langle y, z\rangle{:}\mathsf{Parent}$ and $z{:}\mathsf{Male}$. If z were an individual name, the term 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 (1) 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 (Rousset 1999)). The ordering is important in order to maintain the connection between the rolled up term and the rest of the query. For example, rolling up query (1) in the reverse order would lead to the non-equivalent query

$$\mathsf{Bill}{:}\exists \mathsf{Parent}.\top \wedge y{:}\exists \mathsf{Parent}.\top \wedge z{:}\mathsf{Male}.$$

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

$$\langle x, y\rangle{:}\mathsf{Path} \wedge \langle y, z\rangle{:}\mathsf{Path} \wedge \langle z, x\rangle{:}\mathsf{Path} \qquad (2)$$

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.[4] 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.

For example, if in the query (2) the variable x is substituted by the individual name a, then it can be transformed into the query

$$\langle a, y\rangle{:}\mathsf{Path} \wedge \langle y, z\rangle{:}\mathsf{Path} \wedge z{:}(\exists \mathsf{Path}.P_a),$$

which no longer contains a cycle composed only of variables. Consequently, it can be rolled up into the single concept term

$$a{:}\exists \mathsf{Path}.(\exists \mathsf{Path}.(\exists \mathsf{Path}.P_a))$$

where the concept P_a is used to close the cycle. 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.

In order to deal with variables, one final problem remains to be overcome. 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 $x{:}\exists \mathsf{Parent}.\top$. We cannot simply treat x as an individual and use the standard instantiation technique to reduce the query to KB satisfiability, because x can be interpreted as any element in the domain: in this case we need to verify that the interpretation of the concept $\exists \mathsf{Parent}.\top$ is nonempty in every interpretation that satisfies the KB. However, it is easy to see that 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]

We are now in a position to present a procedure for answering an arbitrary boolean conjunctive query. The first step is to eliminate role terms from the query using the rolling up procedure, with the directed graph induced by the query being used to select an appropriate order in which to apply single rolling up steps. This is done by repeatedly applying one of the following steps until all role terms have been eliminated:

1. If the graph contains a leaf node y (i.e., a node with one incoming edge $\langle x, y\rangle$ and no outgoing edges), then the role term $\langle x, y\rangle{:}R$ is rolled up, and the edge $\langle x, y\rangle$ is removed from the graph.

[4] This is a property of most DLs, and of all those implemented in state of the art systems.

[5] Some earlier DL systems cannot reason with Tbox axioms of this kind (Baader & Hollunder 1991; Bresciani, Franconi, & Tessaris 1995), and this might restrict the kinds of query that could be answered.

2. Otherwise, if the graph contains a confluent node y (i.e., one with multiple incoming edges), then all role terms $\langle x, y\rangle{:}R$ are rolled up, and all edges $\langle x, y\rangle$ are removed from the graph (if y is a variable, then it is first replaced with an individual name chosen nondeterministically from the KB).
3. Finally, if the graph 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 2 above.

The query now contains only concept terms, and evaluates to true iff every term evaluates to true (for some nondeterministic replacement of variables with individual names).

Extensions

For the sake of simplicity, we have so far 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.

DL expressivity

The technique described can be used with a wide range of DL languages. For example, qualified number restrictions, transitive roles and a role hierarchy (Horrocks, Sattler, & Tobies 1999b) could be added to the language without changing the rolling up procedure. Moreover, the efficiency of the rolling up procedure can actually be improved if the language is extended to include inverse roles, i.e., roles of the form R^{-1}, where $(i, j) \in (R^{-1})^{\mathcal{I}}$ iff $(j, i) \in R^{\mathcal{I}}$ (Horrocks & Sattler 1999). With inverse roles the rolling up procedure can be simplified because the orientation of the edges in the graph induced by the query is no longer relevant. For example, the term $\langle \mathsf{John}, \mathsf{Bill}\rangle{:}\mathsf{Brother}$ can be rolled up in either direction to give $\mathsf{John}{:}(\exists \mathsf{Brother}.P_{\mathsf{Bill}})$ or $\mathsf{Bill}{:}(\exists \mathsf{Brother}^{-1}.P_{\mathsf{John}})$. Since the query graph is no longer directed, every connected subgraph without cycles can be treated as a tree and, moreover, each connected component of the graph can be collapsed into a single concept term.

Disjunctive queries

As we have already mentioned, it is possible to extend the basic framework to deal with disjunctions of boolean conjunctive queries, i.e., queries of the form $Q_1 \vee \ldots \vee Q_n$, where each Q_i is a boolean conjunctive query. We will make the assumption that no variable ever occurs in more than one conjunctive query (i.e., the sets of variables occurring in the conjunctive queries are pairwise disjoint).

Even with this simplification, verifying the truth value of a query cannot be achieved by verifying each conjunctive query separately and returning true iff any one of the conjunctive queries evaluates to true. This is because of the potential disjunctive information present in the KB. For example, consider the KB $\langle \emptyset, \{\mathsf{Bill}{:}(\mathsf{PhD} \sqcup \mathsf{MsC})\}\rangle$, and the disjunctive query

$$\mathsf{Bill}{:}\mathsf{PhD} \vee \mathsf{Bill}{:}\mathsf{MsC}.$$

It is easy to see that the query should evaluate to true, but that none of the disjuncts is a logical consequence of the KB. In fact, in order to correctly evaluate the query it is necessary to consider both the terms together, and to test the satisfiability of the KB

$$\langle \emptyset, \{\mathsf{Bill}{:}(\mathsf{PhD} \sqcup \mathsf{MsC}), \mathsf{Bill}{:}\neg\mathsf{PhD}, \mathsf{Bill}{:}\neg\mathsf{MsC}\}\rangle.$$

Clearly, this KB is unsatisfiable, giving the correct answer. A similar situation could arise w.r.t. variables, e.g., with the query $x{:}\mathsf{PhD} \vee y{:}\mathsf{MsC}$. In this case the problem must be reduced to testing the satisfiability of the KB

$$\langle \{\top \sqsubseteq \neg\mathsf{PhD}, \top \sqsubseteq \neg\mathsf{MsC}\}, \{\mathsf{Bill}{:}(\mathsf{PhD} \sqcup \mathsf{MsC})\}\rangle.$$

Again, this KB is clearly unsatisfiable.

The examples given above suggest how the evaluation of disjunctive queries should be performed. The procedure can be summarised in the following three steps.[6] Firstly, each disjunct is transformed into a conjunction of concept terms as per the standard rolling up procedure. Secondly, the disjunction of these conjunctive terms is converted into its conjunctive normal form, the result being a conjunction of disjunctions of concept terms:

$$(q_{1,1} \vee \ldots \vee q_{1,n}) \wedge \ldots \wedge (q_{k,1} \vee \ldots \vee q_{k,n}).$$

Finally, each of the disjunctions of concept terms $q_{i,1} \vee \ldots \vee q_{i,n}$ is separately verified by adding its negation to the KB and testing the unsatisfiability of the result. The procedure returns true (i.e., the original disjunctive query evaluates to true) iff the KB is unsatisfiable in every case.

Discussion

In this paper we have presented a general technique for providing an expressive query language for a DL based knowledge representation system. 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 (Levy & Rousset 1996a; Donini *et al.* 1998; Calvanese, De Giacomo, & Lenzerini 1999). Using Datalog as a query language can provide the ability to formulate recursive queries (Cadoli, Palopoli, & Lenzerini 1997), but on the other hand, the combination with expressive DLs soon leads to undecidability (Levy & Rousset 1996b). 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 (or future) DL system equipped with the KB satisfiability reasoning service.

Our plans for future work include an implementation of the technique on top of the FaCT system (Horrocks 1998),

[6] Full details can be found in (Horrocks *et al.* 1999a).

which has recently been extended to include Abox reasoning (Tessaris & Gough 1998), 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

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

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

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

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

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

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

De Giacomo, G., and Lenzerini, M. 1995. What's in an aggregate: Foundations for description logics with tuples and sets. In *Proc. of IJCAI-95*.

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

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

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

Haarslev, V., and Möller, R. 1999b. RACE system description. In *Proc. of DL'99*, 130–132.

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

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

Horrocks, I.; Sattler, U.; and Tobies, S. 1999b. Practical reasoning for expressive description logics. In *Proc. of LPAR'99*, 161–180. Springer-Verlag.

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

Levy, A. Y., and Rousset, M.-C. 1996a. Carin: A representation language combining horn rules and description logics. In *Proc. of ECAI-96*.

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

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

MacGregor, R. M. 1991. Inside the LOOM description classifier. *SIGART Bulletin* 2(3):88–92.

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

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

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

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

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