Querying the Semantic Web: A Formal Approach

Ian Horrocks and Sergio Tessaris

Department of Computer Science
University of Manchester
Manchester, UK
{horrocks|tessaris}@cs.man.ac.uk

Abstract. Ontologies are set to play a key role in the Semantic Web, and several web ontology languages, like DAML+OIL, are based on DLs. These not only provide a clear semantics to the ontology languages, but allows them to exploit DL systems in order to provide correct and complete reasoning services.

Recent results shown that DL systems can be enriched by a conjunctive query language, providing a solution to one of the weakness of traditional DL systems. These results can be transfered to the Semantic Web community, where the need for expressive query languages is witnessed by different proposals (like DQL for DAML+OIL).

In this paper we present a logical framework for conjunctive query answering in DAML+OIL. Moreover, we provide a sound and complete algorithm based on recent Description Logic research.

1 Introduction

Description Logics (DLs) are a well-known family of knowledge representation formalisms based on the notion of concepts (classes) and roles (properties). DLs have proved useful in wide range of applications including configuration [18], databases [5,7] and ontological engineering (i.e., the design, maintenance and deployment of ontologies).

The use of DLs in ontological engineering has been highlighted by the recent explosion of interest in the Semantic Web [2]. Ontologies are set to play a key role in the Semantic Web, where they will provide a source of shared and precisely defined terms that can be used to describe web resources and improve their accessibility to automated processes [9]. Several prominent web ontology languages, in particular OIL and DAML+OIL [11], are based on DLs; this allows them to exploit formal results (e.g., w.r.t. the decidability and complexity of key inference problems [8]) and algorithm designs from DL research, as well as to use DL based knowledge representation systems to provide reasoning support for web applications [14].

In order to maximise the utility of Web ontologies, it will be necessary not only to reason with ontology classes, but also with individuals (web resources) that instantiate them, and in particular to answer queries over sets of such

I. Horrocks and J. Hendler (Eds.): ISWC 2002, LNCS 2342, pp. 177–191, 2002.

individuals (e.g., see [4]). This highlights a serious shortcoming of many DL based knowledge representation systems: the inadequacy of their query languages. In this paper we show how the query answering technique presented in [17] can be used to provide query answering services for conjunctive query languages such as the one recently proposed for DAML+OIL (see `http://www.daml.org/listarchive/joint-committee/1052.html`).

Recent years have witnessed the transfer of algorithmic techniques used for terminological reasoning (Tbox reasoning) to the development of both algorithms and optimised implementations that also support reasoning about individuals (Abox reasoning), e.g., see [10,12,22]. Although these systems provide sound and complete Abox reasoning for very expressive logics, they often have rather weak Abox query languages. Typically, these only support instantiation (is an individual a an instance of a class C?), realisation (what are the most specific classes a is an instance of?) and retrieval (which individuals are instances of C?). The reason for this weakness is that, in these expressive logics, all reasoning tasks are reduced to that of determining knowledge base (KB) satisfiability. 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 and variables.

In [6] and [17] it is shown that a more sophisticated reduction to KB unsatisfiability can be used for answering conjunctive queries similar to those supported by relational databases.[1] However, with 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.

In this paper we show how, by placing certain restrictions on the use of variables in the query (in particular, their use in query cycles), we can adapt this technique to DAML+OIL. We will also show how some simple extensions can be supported. Completely removing these restrictions causes problems, in particular when variables are used to force cycles in the query. Due to lack of space, these problems are not discussed here; for full technical details the reader is referred to [21].

We will focus on the problem of answering boolean queries, i.e., determining if a query without free variables is true with respect to a KB. Retrieval, i.e., returning the set of all tuples (of individuals) that answer a query, can be turned into a set of boolean queries for all candidate tuples as described in [17]. Clearly this would be extremely inefficient if naively implemented, and we discuss some basic techniques that can be used to improve performance.

2 Preliminaries

2.1 DAML+OIL

DAML+OIL is an ontology language, and as such is designed to describe the *structure* of a domain. DAML+OIL takes an object oriented approach, with the

[1] This is inspired by the use of Abox reasoning to decide conjunctive query containment (see [15,5]).

structure of the domain being described in terms of *classes* and *properties*. An ontology consists of a set of *axioms* that assert, e.g., subsumption relationships between classes or properties. RDF is used to add assertions that resources[2] (pairs of resources) are instances of DAML+OIL classes (properties). When a resource r is an instance of a class C we say that r has type C.

From a formal point of view, DAML+OIL can be seen to be equivalent to the expressive description logic $\mathcal{SHIQ}$ [16] with the addition of existentially defined classes (i.e., the oneOf constructor) and *datatypes* (often called concrete domains in DLs [1]). A DAML+OIL ontology corresponds to a DL terminology (Tbox), and the set of RDF axioms asserting facts about resources corresponds to a DL Abox. As in a DL, DAML+OIL classes can be names (URIs) or *expressions*, and a variety of *constructors* are provided for building class expressions. The expressive power of the language is determined by the class (and property) constructors supported, and by the kinds of axiom supported.

Constructor	DL Syntax	Example
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	Human $\sqcap$ Male
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	Doctor $\sqcup$ Lawyer
complementOf	$\neg C$	$\neg$Male
oneOf	$\{x_1 \ldots x_n\}$	$\{\text{john}, \text{mary}\}$
toClass	$\forall P.C$	$\forall$hasChild.Doctor
hasClass	$\exists P.C$	$\exists$hasChild.Lawyer
hasValue	$\exists P.\{x\}$	$\exists$citizenOf.{USA}
minCardinalityQ	$\geqslant nP.C$	$\geqslant$2hasChild.Lawyer
maxCardinalityQ	$\leqslant nP.C$	$\leqslant$1hasChild.Male
cardinalityQ	$=n\,P.C$	=1 hasParent.Female

Fig. 1. DAML+OIL class constructors

Figure 1 summarises the constructors supported by DAML+OIL. The standard DL syntax is used for compactness as the RDF syntax is rather verbose. In the RDF syntax, for example, Human $\sqcap$ Male would be written as

```
<daml:Class>
  <daml:intersectionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Human"/>
    <daml:Class rdf:about="#Male"/>
  </daml:intersectionOf>
</daml:Class>
```

[2] Everything describable by RDF is called a resource. A resource could be Web accessible, e.g., a Web page or part of a Web page, but it could also be an object that is not directly accessible via the Web, e.g., a person. Resources are named by URIs plus optional anchor ids. See `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/` for more details.

while $\geqslant$2hasChild.Lawyer would be written as

```
<daml:Restriction daml:minCardinalityQ="2">
  <daml:onProperty rdf:resource="#hasChild"/>
  <daml:hasClassQ rdf:resource="#Lawyer"/>
</daml:Restriction>
```

The meaning of the first three constructors (intersectionOf, unionOf and complementOf) is relatively self-explanatory: they are just the standard boolean operators that allow classes to be formed from the intersection, union and negation of other classes. The oneOf constructor allows classes to be defined existentially, i.e., by enumerating their members.

The toClass and hasClass constructors correspond to slot constraints in a frame-based language and to value and existential restrictions in a DL. The class $\forall P.C$ is the class all of whose instances are related via the property P only to resources of type C, while the class $\exists P.C$ is the class all of whose instances are related via the property P to at least one resource of type C. The hasValue constructor is just shorthand for a combination of hasValue and oneOf.

The minCardinalityQ, maxCardinalityQ and cardinalityQ constructors (known in DLs as qualified number restrictions) are generalisations of the hasClass and hasValue constructors. The class $\geqslant nP.C$ ($\leqslant nP.C$, $=n\,P.C$) is the class all of whose instances are related via the property P to at least (at most, exactly) n *different* resources of type C. The emphasis on different is because there is no unique name assumption with respect to resource names (URIs): it is possible that many URIs could name the same resource.

Note that arbitrarily complex nesting of constructors is possible. Moreover, XML Schema *datatypes* (e.g., so called primitive datatypes such as strings, decimal or float, as well as more complex derived datatypes such as integer subranges) can be used anywhere that a class name might appear.

The formal semantics of the class constructors is given by DAML+OIL's model-theoretic semantics[3] or can be derived from the specification of a suitably expressive DL (e.g., see [14]).

As already mentioned, besides the set of constructors supported, the other aspect of a language that determines its expressive power is the kinds of axiom supported. Figure 2 summarises the axioms supported by DAML+OIL. These axioms make it possible to assert subsumption or equivalence with respect to classes or properties, the disjointness of classes, the equivalence or non-equivalence of individuals (resources), and various properties of properties.

Note that all of the class and individual axioms, as well as the uniqueProperty and unambiguousProperty axioms, can be reduced to subClassOf and sameClassAs axioms (as can be seen from the DL syntax). In fact sameClassAs could also be reduced to subClassOf as a sameClassAs axiom $C \equiv D$ is equivalent to a pair of subClassOf axioms, $C \sqsubseteq D$ and $D \sqsubseteq C$.

As we have seen, DAML+OIL allows properties of properties to be asserted. It is possible to assert that a property is unique (i.e., functional) and unambigu-

[3] `http://www.w3.org/TR/daml+oil-model`

Axiom	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
sameClassAs	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
samePropertyAs	$P_1 \equiv P_2$	cost $\equiv$ price
disjointWith	$C_1 \sqsubseteq \neg C_2$	Male $\sqsubseteq$ $\neg$Female
sameIndividualAs	$\{x_1\} \equiv \{x_2\}$	{President_Bush} $\equiv$ {G_W_Bush}
differentIndividualFrom	$\{x_1\} \sqsubseteq \neg\{x_2\}$	{john} $\sqsubseteq$ $\neg${peter}
inverseOf	$P_1 \equiv P_2^-$	hasChild $\equiv$ hasParent$^-$
transitiveProperty	$P^+ \sqsubseteq P$	ancestor$^+$ $\sqsubseteq$ ancestor
uniqueProperty	$\top \sqsubseteq \leqslant 1P$	$\top \sqsubseteq \leqslant$1hasMother
unambiguousProperty	$\top \sqsubseteq \leqslant 1P^-$	$\top \sqsubseteq \leqslant$1isMotherOf$^-$

Fig. 2. DAML+OIL axioms

ous (i.e., its inverse is functional). It is also possible to use inverse properties and to assert that a property is transitive.

2.2 Description Logic

In this paper we concentrate on a DL less expressive than DAML+OIL, since answering to conjunctive queries over the complete DAML+OIL is still an open problem.

$\mathcal{SHIQ}$ is built over a signature of distinct sets of concept ($\mathcal{CN}$), role ($\mathcal{RN}$) and individual ($\mathcal{O}$) names. In addition, we distinguish two non-overlapping subsets of $\mathcal{RN}$ ($\mathcal{TRN}$ and $\mathcal{FRN}$) which denote the transitive and the functional roles. The set of all $\mathcal{SHIQ}$ roles is equal to the set of role names $\mathcal{RN}$ union the set of the inverse roles $\{R^- \mid P \in \mathcal{RN}\}$. The set of all $\mathcal{SHIQ}$ concepts is the smallest set such that every concept name in $\mathcal{CN}$ and the symbols $\top$, $\bot$ are concepts, and if C,D are concepts, R is a role, and n an integer, then $\neg C$, $(C \sqcap D)$, $(C \sqcup D)$, $(\forall R.C)$, $(\exists R.C)$, $\geqslant nR.C$, and $\leqslant nR.C$ are concepts.

An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a nonempty domain $\Delta^{\mathcal{I}}$ and a interpretation function $\cdot^{\mathcal{I}}$. The interpretation function maps concepts into subsets of $\Delta^{\mathcal{I}}$, individual names into elements of $\Delta^{\mathcal{I}}$, and role names into subsets of the cartesian product of $\Delta^{\mathcal{I}}$ ($\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$). Concept names are interpreted as subsets of $\Delta^{\mathcal{I}}$, while complex expressions are interpreted according to the following equations (see [19])

$$
\begin{aligned}
\top^{\mathcal{I}} = \Delta^{\mathcal{I}} \ (C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
\bot^{\mathcal{I}} = \emptyset \quad (C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
\neg C^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \left\{x \in \Delta^{\mathcal{I}} \mid \forall y (x,y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\right\} \\
(\exists R.C)^{\mathcal{I}} &= \left\{x \in \Delta^{\mathcal{I}} \mid \exists y (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\right\} \\
(\geqslant nR.C)^{\mathcal{I}} &= \left\{x \in \Delta^{\mathcal{I}} \mid \sharp\left\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\right\} \geq n\right\} \\
(\leqslant nR.C)^{\mathcal{I}} &= \left\{x \in \Delta^{\mathcal{I}} \mid \sharp\left\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\right\} \leq n\right\}
\end{aligned}
$$

A role and its inverse must be interpreted according to the equation

$$R^{-\mathcal{I}} = \left\{(x,y) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (y,x) \in R^{\mathcal{I}}\right\}.$$

In addition, the interpretation function must satisfy the transitive and functional restrictions on role names; i.e. for any $R \in \mathcal{TRN}$ if $(x,y) \in R^{\mathcal{I}}$ and $(y,z) \in R^{\mathcal{I}}$, then $(x,z) \in R^{\mathcal{I}}$, and for any $F \in \mathcal{FRN}$ if $(x,y) \in F^{\mathcal{I}}$ and $(x,z) \in F^{\mathcal{I}}$, then $y = z$.

The semantics of DL 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 $a, b \in \mathcal{O}$ 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.

2.3 Knowledge Bases

A $\mathcal{SHIQ}$ knowledge base $\mathbf{K}$ is a finite set of statements of the form:

$$C \sqsubseteq D, R \sqsubseteq S, a{:}C, \langle a,b\rangle{:}R$$

where C, D are $\mathcal{SHIQ}$ concepts, R, S roles, and a, b individual names. The first two kinds of statement are called *terminological*, while the two latter ones are called *assertional*. Intuitively, terminological statements describe intensional properties of all the elements of the domain, while assertional statements assign properties of some named elements.

We say that an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ satisfies the terminological statement $C \sqsubseteq D$ ($R \sqsubseteq S$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ ($R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$), and the assertional statement $a{:}C$ ($\langle a,b\rangle{:}R$) iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ ($(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$). When an interpretation $\mathcal{I}$ satisfies a statement α, we use the notation $\mathcal{I} \models \alpha$. An interpretation $\mathcal{I}$ satisfies (or is a model for) a KB $\mathbf{K}$ iff it satisfies all the statements in $\mathbf{K}$ (written as $\mathcal{I} \models \mathbf{K}$).

2.4 Query Language

Query answering services provided by a DL system can be seen as the process of verifying whether a given statement (the query) is a logical consequence of the knowledge base (written as $\mathbf{K} \models \alpha$). The meaning of logical consequence is given in terms of interpretations; i.e. a statement is logical consequence of a KB $\mathbf{K}$ if it is satisfied in every interpretation satisfying $\mathbf{K}$ ($\mathbf{K} \models \alpha$ iff for any interpretation $\mathcal{I}$, $\mathcal{I} \models \mathbf{K}$ implies $\mathcal{I} \models \alpha$). For example, instantiation can be written as $\mathbf{K} \models a{:}C$ (i.e., a is an instance of C in every model of $\mathbf{K}$).

Using the same mechanism, we extend the kind of queries we can ask by introducing a conjunctive query language whose terms are assertional statements (see [17]). For this purpose we consider a set of variable names $\mathcal{V}$ distinct from the individual names ($\mathcal{O}$). Analogously to conjunctive queries in the database setting, variables can be used in place of individuals and are considered as existentially quantified.

A DL *boolean conjunctive query* is defined as a conjunction of terms of the form $x{:}C$ or $\langle x, y\rangle{:}R$, where C is a concept, R is a role, and x, y are variable or individual names taken from $\mathcal{V} \cup \mathcal{O}$. We call the first kind *concept terms* and the latter kind *role terms*.

The semantics of a boolean conjunctive query follows the schema shown above for the knowledge bases. The difference is that we need to consider the variable names, since the satisfiability of a term may be affected by the assignment of the variables. Given an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, we consider *evaluations* defined as mappings from names in $\mathcal{V} \cup \mathcal{O}$ to elements of the interpretation domain $\Delta^{\mathcal{I}}$ (with the constraint that evaluations must agree with the interpretation function on the mapping of individual names). We say that the interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ satisfies the term $x{:}C$ ($\langle x, y\rangle{:}R$) w.r.t. an evaluation ν, written as $\mathcal{I} \models_{\nu} x{:}C$ ($\mathcal{I} \models_{\nu} \langle x, y\rangle{:}R$), iff $\nu(x) \in C^{\mathcal{I}}$ ($\langle \nu(x), \nu(y)\rangle \in R^{\mathcal{I}}$). This is extended to arbitrary conjunctive queries: an interpretation $\mathcal{I}$ satisfies the conjunctive query $q = t_1 \wedge \ldots \wedge t_n$ w.r.t. an evaluation ν iff $\mathcal{I} \models_{\nu} t_i$ for every $i = 1, \ldots, n$.

Note that we do not require that variables are interpreted as the individual names appearing in the KB; instead they can be mapped to arbitrary elements of the interpretation domain. For example, let us consider the KB containing only the assertion $\texttt{sam}{:}\exists\texttt{Has\_child}.\texttt{FEMALE}$, and the query $\langle \texttt{sam}, y\rangle{:}\texttt{Has\_child}$. If we restrict variables to individual names only, then the query is not a logical consequence of the KB, because there is no individual name asserted to be related to sam. If we allow variables to range over arbitrary elements of interpretation domains, then the query is a logical consequence of the KB. This can be seen by considering that the query is equivalent to the query $\texttt{sam}{:}\exists\texttt{Has\_child}.\top$, and that the concept $\exists\texttt{Has\_child}.\top$ is more general than $\exists\texttt{Has\_child}.\texttt{FEMALE}$.

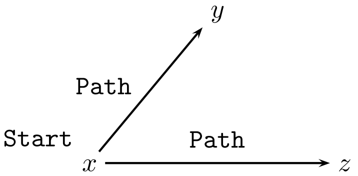
In answering boolean queries, we are not really interested in the evaluation itself but only on the satisfiability of the given query; we say that $\mathcal{I}$ satisfies the query q (written $\mathcal{I} \models q$) iff there is an evaluation ν such that $\mathcal{I} \models_{\nu} q$.

Query graph To present the query answering algorithm we associate a *query graph* to each conjunctive query. The main idea is to consider a conjunctive query as a directed graph, where the nodes are variable and individual names. In addition, concept and role terms provide labels for nodes and edges respectively.

For example, the query

$$x{:}\texttt{Start} \wedge \langle x, y\rangle{:}\texttt{Path} \wedge \langle x, z\rangle{:}\texttt{Path}$$

corresponds to the graph



In this paper we restrict to queries whose graphs are acyclic (regardless the direction of the edges). For example, adding the term $\langle y, z\rangle{:}\texttt{Path}$ to the previous query:

$$\{x{:}\texttt{Start}, \langle x, y\rangle{:}\texttt{Path}, \langle y, z\rangle{:}\texttt{Path}, \langle x, z\rangle{:}\texttt{Path}\}$$

makes the corresponding graph cyclical. This restriction leads to a much more efficient procedure, and the algorithm still works with very expressive DLs.

There is ongoing research to extend the algorithm to arbitrary shaped queries, and expressive DLs. Encouraging results have been published for DLs less expressive than DAML+OIL (see [6,17,20]).

Query retrieval Using the definition of boolean queries we can easily extend the formalism to retrieve arbitrary tuples of individual names. We use the notation $\langle x_1, \ldots, x_n\rangle \leftarrow q$ to indicate that variables $x_1, \ldots, x_n$ appearing in q must be bound to individual names, and constitute the answer to the query. We call these variables *distinguished.*

Formally, the *answer set* of a query $\langle x_1, \ldots, x_n\rangle \leftarrow q$ w.r.t. the KB $\mathbf{K}$ is the set n-ary tuples defined by

$$\{\langle a_1, \ldots, a_n\rangle \in \mathcal{O}^n \mid \mathbf{K} \models q[x_1/a_1, \ldots, x_n/a_n]\}\,,$$

where $q[x/a]$ indicates the query q with all the occurrences of variable x substituted by the individual name a.

3 Answering Boolean Queries

In this section we show how to answer to boolean queries (see Section 2.4); i.e. queries not returning set of tuples but only a yes/no answer. In Section 4 we show that using this algorithm we can provide answers to non-boolean queries as well.

A boolean query can be partitioned in one or more connected components by considering its query graph. Unconnected components do not share variables, therefore they can be considered independently to each other.

For example, the query

$$\langle\texttt{Mary}, y\rangle{:}\texttt{children} \wedge y{:}\texttt{MALE} \wedge z{:}\texttt{STUDENT}$$

has two connected components: ($\langle\texttt{Mary}, y\rangle{:}\texttt{children} \wedge y{:}\texttt{MALE}$) and ($z{:}\texttt{STUDENT}$). Since they not share any variable, the query is a logical consequence of a KB iff the two components $\langle\texttt{Mary}, y\rangle{:}\texttt{children} \wedge y{:}\texttt{MALE}$ and $z{:}\texttt{STUDENT}$ are logical consequence of the KB.

Boolean query answering, i.e. 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,

$$\{\texttt{STUDENT} \sqsubseteq \texttt{PERSON}, \texttt{Tom:STUDENT}\} \models \texttt{Tom:PERSON}$$

iff the KB

$$\{\texttt{STUDENT} \sqsubseteq \texttt{PERSON}, \texttt{Tom:STUDENT}, \texttt{Tom:}\neg\texttt{PERSON}\}$$

is not satisfiable.

This is true not only for individual names, but for variables as well. The query x:PERSON is satisfied iff in every model of the KB the interpretation of PERSON is not the empty set. This can be verified by checking whether the KB plus the axiom PERSON $\sqsubseteq \bot$ is satisfiable. If this is the case, then there is at least a model of the KB in which the interpretation of PERSON is the empty set ($\bot$ is by definition the empty set).

This simple approach cannot be used in our case since a query may also contain role terms. However, the idea is to transform the initial query into an equivalent query containing only a single concept term (see [5,17]). In terms of query graphs this means collapsing the DAG into a single node, by eliminating all the edges.

Firstly, we consider queries containing only variable names; then we show that constants (i.e. individual names) can be handled in a similar fashion.

3.1 Queries without Constants

Let us consider the simple query $\langle y, z\rangle$:children $\wedge$ z:MALE. The query is satisfied if there is an element (y), related by role children to an element (z) of the class MALE.[4] Given the semantics of DL operators (see Section 2.2), the same query can be paraphrased as the single term y:$\exists$children.MALE.

The intuition from the example is substantiated by the fact that the query corresponds to the first order logic formula $\exists y \exists z$(children(y, z) $\wedge$ MALE(z)), which is the first order logic translation of the term y:$\exists$children.MALE (see [3]). We indicate the transformation of query formulae suggested by this example, as the *rolling-up* of role terms.

Inverse role constructor (e.g. children$^-$) enables the rolling-up in both directions. In fact, the role term in the example can be rolled-up into the variable z obtaining the query z:$\exists$children$^-$.$\top$ $\wedge$ z:MALE.

Note that the transformation eliminates one of the variables (z in the example); therefore the equivalence is guaranteed iff the variable being eliminated does not appear anywhere else in the query.

Let us consider the query $\langle x, y\rangle$:children $\wedge$ $\langle y, z\rangle$:has_degree, and the KB containing the assertion Mary:($\exists$children.MALE $\sqcap$ $\exists$has_degree.PHD). This query is not a logical consequence of the given KB, because there is nothing in the KB forcing the role chain expressed by the query. A careless use of rolling-up, applied to the first role term, produces the query x:$\exists$children.$\top$$\wedge$$\langle y, z\rangle$:has_degree. The resulting query is a logical consequence of the KB; therefore this transformation does not guarantee correctness.

The problem highlighted by this example can be overcome by eliminating variables appearing in a single role term. Multiple concept terms (like x:MALE $\wedge$ x:PIG) are not a problem; since they can be collapsed into a single one by using the conjunction construct (e.g. x:(MALE $\sqcap$ PIG)). The assumption that the query graph is a DAG (see Section 2.4) ensures that there is always at least a variable appearing in one role term only.

[4] This must be true in every interpretation satisfying the KB being queried.

3.2 Queries with Constants

The rolling-up described in the last section cannot be used as it is when there are constants (i.e. individual names) in the query. The reason for this is that names are significant, so we cannot treat them as variables.

Let us consider the example of the previous section where we substitute variable z with an individual name:

$$\langle y, \mathtt{Bill}\rangle{:}\mathtt{children} \wedge \mathtt{Bill}{:}\mathtt{MALE}.$$

This query is not a logical consequence of a KB containing only the assertion $\mathtt{Bill}{:}\mathtt{MALE} \sqcap \exists\mathtt{children}.\mathtt{MALE}$; because the role term $\langle y, \mathtt{Bill}\rangle{:}\mathtt{children}$ is not satisfied in every model of the KB. However, if we roll-up the role term, ignoring the fact that $\mathtt{Bill}$ is a constant, we obtain the query $y{:}\exists\mathtt{children}.\mathtt{MALE}$, which is a logical consequence of the assertion in the KB.

The problem can be solved by using the *one-of* DL construct, which enables to describe a concept by enumerating its members. For example, the interpretation of the concept $\{\ \mathtt{Sally}, \mathtt{Bill}\ \}$ is the set containing the elements corresponding to $\mathtt{Sally}$ and $\mathtt{Bill}$ (see Section 2.2). It is not difficult to realise that a query term like $\mathtt{Bill}{:}\mathtt{MALE}$ is equivalent to $z{:}\{\ \mathtt{Bill}\ \} \wedge z{:}\mathtt{MALE}$; where z is a newly introduced variable. In fact, the term $z{:}\{\ \mathtt{Bill}\ \}$ guarantees that variable z is always interpreted as the constant $\mathtt{Bill}$.

Generalising this idea, we can remove all the constants from the query by introducing appropriate concept terms involving the *one-of* DL construct. For example, the query example is transformed into

$$\langle y, z\rangle{:}\mathtt{children} \wedge z{:}\mathtt{MALE} \wedge z{:}\{\ \mathtt{Bill}\ \},$$

by replacing all the occurrences of $\mathtt{Bill}$ with the new variable z, and introducing the new term $z{:}\{\ \mathtt{Bill}\ \}$. Now the query can be rolled-up as described in Section 3.1, obtaining the concept term $y{:}\exists\mathtt{children}.(\mathtt{MALE} \sqcap \{\ \mathtt{Bill}\ \})$.

Note that it is not necessary to use same variable name for all the occurrences of a constant. The crucial point is that they all have to be constraint by a concept term involving the *one-of* construct. The query example can be transformed into the equivalent

$$\langle y, z'\rangle{:}\mathtt{children} \wedge z'{:}\{\ \mathtt{Bill}\ \} \wedge z{:}\mathtt{MALE} \wedge z{:}\{\ \mathtt{Bill}\ \},$$

since the terms $z'{:}\{\ \mathtt{Bill}\ \}$ and $z{:}\{\ \mathtt{Bill}\ \}$ ensure that z' and z are always interpreted as the same element.

Unfortunately, the DL systems used to support reasoning in DAML+OIL do not provide the *one-of* construct. However, in our case we do not need the full expressivity of *one-of*, and it can be simulated by primitive concept names. 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., Bill:$P_{\mathtt{Bill}}$).

Representative concepts can be used instead of *one-of* to transform the queries. However, they cannot be used to eliminate *one-of* operators from KBs because a representative concept can have instances other than the individual which it represents (i.e., ${P_a}^{\mathcal{I}} \supseteq \{a^{\mathcal{I}}\}$).

4 Retrieving Answer Sets

Ideally, we would like to provide an efficient bottom up procedure for retrieving answer sets. However, in the context of expressive DLs this is not easily achievable, and we are not aware of any lead towards a solution.

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 the use of an approach based on minimal models infeasible. In fact, query answering in the DL setting requires the same reasoning machinery as logical derivation. To use model checking techniques for query answering we must be able to associate a "preferred" model to a given KBs, and this is quite difficult for arbitrary DL KBs.

Let us consider for example a simple KB containing the single axiom Elephant $\sqsubseteq \neg$Mouse, stating that elephants and mice are disjoint, and the Abox assertion hathi:(Elephant $\sqcup$ Mouse). We can identify two minimal (w.r.t. inclusion) interpretations satisfying the KB: in the first the element mapped from the individual hathi is in the extension of the concept Elephant, while in the second it is in the extension of the concept Mouse. Which of the two interpretations can be considered the "preferred" one? The point is that there is not any general mechanism for choosing one, even with this trivial KB.

From the definition of answer set, given in Section 2.4, we can easily derive an algorithm for retrieving tuples of individuals answering a given query. In fact, using the boolean query answering algorithm applied to the query obtained by substituting the distinguished variables with constants, we can test the membership of a given tuple to the answer set. The idea is to iterate among all the possible assignment of the distinguished variables, and checking whether the corresponding tuples belong to the answer set.

Although this procedure is possibly not the most practical one,[5] it fits nicely with the recent proposal for the DQL DAML+OIL query language.[6] In fact, in the proposal a response to a query would consist of a single binding for the distinguished variables, and a "server continuation" which can be used to obtain the next answers (bindings).

[5] The naive evaluation of such a retrieval could be prohibitively expensive, but as we point out in Section 5 it is amenable to optimisation.

[6] The so called DQL query language, discussed in the joint-committee DAML mailing list (see http://www.daml.org/listarchive/joint-committee/1052.html).

Another feature of the above mentioned proposal is the possibility of returning partial bindings when the KB entails the existence of individuals, but those are not among the known names (i.e. not in the set of $\mathcal{O}$).

For example, a KB containing only the assertion $\texttt{Red}{:}\exists\texttt{colour}^{-}.\top$ implies the existence of an element related to `Red` via the role `colour`. However, there is not any individual name which is asserted to correspond to this element. Any query like $\langle x,y\rangle \leftarrow \langle x,y\rangle{:}\texttt{colour}$, with x, y distinguished variables, would not return any answer (i.e. the empty set).

In the proposal is suggested that in such a case an answer would be a binding only for the variable y (the individual name `Red`), while x would be left unspecified (or a bind to a newly invented name representing an anonymous element).

This effect can be achieved in our framework by relaxing the conditions on the variables; i.e. making part of the distinguished variables no longer distinguished. For example, the query $\langle x,y\rangle \leftarrow \langle x,y\rangle{:}\texttt{colour}$ can be relaxed into the query $\langle y\rangle \leftarrow \langle x,y\rangle{:}\texttt{colour}$, where x is no longer distinguished.

In our view, the task of relaxing the conditions should not be incorporated into the basic query answering mechanism, but left to an external layer. For example, this external layer would first tries to answer to the query as it is (leaving all the distinguished variables). If with these restrictions no answers can be retrieved, then different queries can be generated by making one or more variables non-distinguished. The process would continue until an answer is found, and returned to the user.

Several heuristics and ordering can be adopted for the selection of distinguished variables to be relaxed. We think that this mainly depends on the specifications of DQL, which is still an ongoing project. However, the main point is that our logical framework can be used to capture this feature.

5 Speeding Up the Answer

The rolling-up procedure is polynomial in the size of the query, and the KB satisfiability test is EXPTIME for the DL $\mathcal{SHIQ}$. Given the fact that boolean query answering is at least as expensive as KB satisfiability,[7] our algorithm is optimal w.r.t. the class of boolean acyclic conjunctive queries (assuming that the KB satisfiability test is optimal). However, we can use several heuristics to obtain a better behaviour in most of the cases (i.e. practical tractability).

We have empirical evidence that axioms in the KB are one of the major cause of practical intractability (see [13]). As seen in Section 3, the query need to be encoded as an axiom only if is rolled-up into a variable. When the query is rolled-up into an individual name, the query can be transformed into an Abox assertion. Therefore, the choice of node into which a query graph is rolled-up can be used to speed up the KB satisfiability test.

[7] A KB is unsatisfiable iff the query $x{:}\bot$ is a logical consequence of the KB.

Different optimizations can be directed to minimize the choice of individual name candidates for distinguished variables (see Section 4). In fact, in case of query retrieval of n-ary tuples we potentially have to test every possible element of $\mathcal{O}^n$. For reducing the number of candidate individuals for a variable name, we envisage two different techniques. The first one relies on the standard retrieval service provided by DL reasoners (i.e. retrieving all the individual names instances of a given concept), while the second on the structure induced by role terms in the query.

In our setting, the rolling-up is a cheap operation so we can use it to prune the number of candidates. The idea is to roll-up the query into a distinguished variable prior to substitute it with any individual name. The concept we obtain describes necessary conditions for the individuals that can be substituted to this distinguished variable. The concept is used to retrieve the list of individual names being instance of the concept, and the retrieved individuals are the candidates for the distinguished variable.

This technique is not an alternative to the boolean query answering, since tuples membership to the answer set still need to be verified by a boolean query answer. However, this may significantly reduce the number of boolean queries that need to be tested. Moreover, DL systems are usually optimised for retrieval, by means of internal indexes and specialised algorithms (see [10]).

The structure of role terms in the query (i.e. the "shape" of the query graph) can be used to reduce the number of candidates for distinguished variables. This idea is based on the observation that role assertions in the KB do not allow to express incomplete information as the concept assertions do. In fact, role assertions are usually limited to simple statements like $\langle a, b \rangle{:}R$. The crucial point is that two individual names can be related by a role only if there is a role assertion between them. Note that in expressive DLs, like $\mathcal{SHIQ}$, the names of the roles in the query and in the assertions do not need to match. This limited expressivity for roles is shared by most of the DLs studied and/or implemented. Note that this is no longer valid for languages including the *one-of* operator, like DAML+OIL, or without the unique name assumption (see Section 2.2). Therefore, this kind of optimizations need to be used with extreme caution.

For example, if the underlying DL language allows the use of this optimization, and the query contains the role term $\langle x, y \rangle$:`children` (with both x, y distinguished variables), then we can restrict their candidates to pair of individual names having an asserted role between them.

6 Conclusions

In this paper we have described a basic conjunctive query language for DAML+OIL (or any other description logic based ontology language), and presented a formal framework that precisely defines the meaning of such qeries. Moreover, we have shown how queries can be rewritten so that query answering is reduced to the problem of knowledge base satisfiability for the logic corresponding to the ontology language. This enables us to answer queries using standard reasoning techniques, and to guarantee that query answers will be sound and complete in the case that that our knowledge base satisfiability test is sound and complete. The idea is to use implemented description logic systems (or any other system capable of deciding knowledge base satisfiability) to provide sound and complete answers to queries.

There have been a number of proposals for query languages for RDF and DAML+OIL (a review of some of them can be found at `http://139.91.183.30:9090/RDF/publications/state.html`). However, the aproach we have described is, to the best of our knowledge, unique in formalising the problem and in describing a mechanism whereby sound and complete answers to non-trivial queries can be computed using an inference procedure.

The query rewriting relies on a restriction with respect to the use of variables and constants in query expressions, notably that no cyclical references are allowed. Relaxing this condition so that distinguished variables and constants can occur in query cycles is not too difficult, but dealing with non-distinguished variables ocurring in query cycles is still an open research problem (although the problem has been solved for less expressive description logics [17]).

It also remains to be seen how effective such techniques will be in practice. If implemented naively, it is clear that they would be extremely inefficient. However, as we have seen in Section 5, there are many possibilities for optimising implementations in order to speed up query answering. Moreover, this technique lends itself naturally to incremental query answering, where the system can return partial answers without having to wait until the complete answer has been computed.

References

1. Franz Baader and Philipp Hanschke. A schema for integrating concrete domains into concept languages. In *Proc. of IJCAI-91*, pages 452–457, 1991.
2. Tim Berners-Lee. *Weaving the Web.* Harpur, San Francisco, 1999.
3. Alexander Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1–2):353–367, 1996.
4. J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: An architecture for storing and querying RDF data and schema information. In H. Lieberman D. Fensel, J. Hendler and W. Wahlster, editors, *Semantics for the WWW.* MIT Press, 2001.
5. Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the decidability of query containment under constraints. In *Proc. of PODS-98*, pages 149–158, 1998.
6. Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Answering queries using views over description logics knowledge bases. In *Proc. of AAAI 2000*, pages 386–391, 2000.
7. Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. Description logics for conceptual data modeling. In Jan Chomicki and Günter Saake, editors, *Logics for Databases and Information Systems*, pages 229–264. Kluwer Academic Publisher, 1998.
8. Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. The complexity of concept languages. *Information and Computation*, 134:1–58, 1997.

9. Dieter Fensel, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, and Peter F. Patel-Schneider. OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.
10. Volker Haarslev and Ralf Möller. RACER system description. In *Proc. of IJCAR-01*, 2001.
11. I. Horrocks and P. Patel-Schneider. The generation of DAML+OIL. In *Proc. of DL 2001*, pages 30–35, 2001.
12. I. Horrocks, U. Sattler, and S. Tobies. Reasoning with individuals for the description logic $\mathcal{SHIQ}$. In David MacAllester, editor, *Proc. of CADE-2000*, number 1831 in LNCS, pages 482–496. Springer-Verlag, 2000.
13. Ian Horrocks and Peter F. Patel-Schneider. Optimizing description logic subsumption. *J. of Logic and Computation*, 9(3):267–293, 1999.
14. Ian Horrocks and Ulrike Sattler. Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In *Proc. of IJCAI-01*. Morgan Kaufmann, 2001.
15. Ian Horrocks, Ulrike Sattler, Sergio Tessaris, and Stephan Tobies. How to decide query containment under constraints using a description logic. In *Logic for Programming and Automated Reasoning (LPAR 2000)*, volume 1955 of *LNCS*, pages 326–343. Springer, 2000.
16. Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In Harald Ganzinger, David McAllester, and Andrei Voronkov, editors, *Proc. of LPAR'99*, number 1705 in LNAI, pages 161–180. Springer-Verlag, 1999.
17. Ian Horrocks and Sergio Tessaris. A conjunctive query language for description logic aboxes. In *Proc. of AAAI 2000*, pages 399–404, 2000.
18. Deborah L. McGuinness and Jon R. Wright. An industrial strength description logic-based configuration platform. *IEEE Intelligent Systems*, pages 69–77, 1998.
19. Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.
20. Sergio Tessaris. Querying expressive dls. In *Proc. of DL 2001*, 2001.
21. Sergio Tessaris. *Questions and answers: reasoning and querying in Description Logic*. PhD thesis, University of Manchester, 2001.
22. Sergio Tessaris and Graham Gough. Abox reasoning with transitive roles and axioms. In *Proc. of DL'99*, 1999.