

Handling Cyclic Conjunctive Queries

Birte Glimm* and Ian Horrocks
University of Manchester
Manchester, UK
[glimm,horrocks]@cs.man.ac.uk

Abstract

Query containment for conjunctive queries is the problem of checking whether a query q is contained in another query q' with respect to a given Description Logic schema \mathcal{S} . It is known that query containment can also be used to answer queries. Query containment is important in many areas, including information integration, query optimisation, and reasoning about Entity-Relationship diagrams, while query languages that allow the use of variables and individuals in a query, such as conjunctive queries, become a more and more important topic in the area of ontology development and the Semantic Web. We introduce the existing algorithms to decide query containment resp. query answering, and show by means of an example that these algorithms are incomplete for some cyclic queries. We highlight the sources of incompleteness that are easily overlooked in the development of suitable algorithms and suggest an extended algorithm to overcome these problems.

1 Introduction

The Semantic Web [2] aims at making Web resources more accessible to automated processes by augmenting web pages with descriptions of their content. Ontologies are used to provide precisely specified meanings for the descriptions, and with the Web Ontology Language OWL [1] a standardised ontology building language is available. A notable feature of OWL is that two of the three OWL species (OWL Lite and OWL DL) correspond to Description Logics (DLs) [8]. Therefore, existing DL reasoners [6, 7, 11] can be used for automated reasoning about OWL ontologies. Recently, a decision procedure for the DL *SHOIQ* has been introduced [9] that will enable the implementation of reasoning procedures

*This work was supported by an EPSRC studentship.

for the full expressivity of OWL DL. The development of sound and complete query answering algorithms for expressive query languages is, however, still an outstanding issue.

In this paper we highlight some of the difficulties that arise during the development of query answering algorithms and show that the existing approaches are, despite the provided proofs, not yet complete. To determine query answers, we use *query containment* algorithms. This is a well known technique in the database community [5]. Query containment is the problem of checking whether a query q is contained in a query q' w.r.t. a DL schema or TBox. Query containment has, for example, proved useful in the context of information integration, query optimisation, and data warehousing. However, the query containment algorithms suffer from the same incompleteness problems that we show for the case of query answering.

The next section provides a short overview of the Description Logic \mathcal{DLR}_{reg} and introduces conjunctive queries. Section 3 shows how the query containment algorithm works and provides an incompleteness example for the algorithm introduced by Calvanese et al. [3] and adapted by Horrocks et al. [10]. We then describe an envisaged solution to overcome this shortcoming.

2 Preliminaries

Before explaining the query containment algorithm, we give a short introduction to the DL \mathcal{DLR}_{reg} [4] used by Calvanese et al. for their proposed query containment algorithm [3]. \mathcal{DLR}_{reg} is an expressive DL suitable, for example, for reasoning about Entity-Relationship (ER) diagrams.

2.1 Syntax and Semantics of \mathcal{DLR}_{reg}

In contrast to many other DLs, \mathcal{DLR}_{reg} provides n-ary relations as well as binary relations and regular expressions. If \mathbf{A} is an atomic concept, \mathbf{C} a concept expression, \mathbf{p} an atomic relation, \mathbf{r} an arbitrary relation, \mathbf{e} a regular expression, n , k , i , and j non-negative integers, where n denotes the arity of a relation and i, j denote components of relations, then \mathcal{DLR}_{reg} expressions are built according to the following syntax:

$$\begin{aligned} \mathbf{r} &::= \top_n \mid \mathbf{p} \mid (\$i/n: \mathbf{C}) \mid \neg \mathbf{r} \mid \mathbf{r}_1 \sqcap \mathbf{r}_2 \\ \mathbf{e} &::= \epsilon \mid \mathbf{r}_{\$i, \$j} \mid \mathbf{e}_1 \circ \mathbf{e}_2 \mid \mathbf{e}_1 \sqcup \mathbf{e}_2 \mid \mathbf{e}^* \\ \mathbf{C} &::= \top \mid \mathbf{A} \mid \neg \mathbf{C} \mid \mathbf{C}_1 \sqcap \mathbf{C}_2 \mid \exists \mathbf{e}. \mathbf{C} \mid \exists [\$i] \mathbf{r} \mid \leq k [\$i] \mathbf{r} \end{aligned}$$

The function $arity(\mathbf{r})$ returns the arity of the relation \mathbf{r} and for all relations \mathbf{r} the i, j have to be smaller or equal to $arity(\mathbf{r})$. We use the usual abbreviations such as $\mathbf{C} \sqcup \mathbf{D}$ for $\neg(\neg \mathbf{C} \sqcap \neg \mathbf{D})$. The semantics are given by an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, which consists of a non-empty domain of interpretation $\Delta^{\mathcal{I}}$ and an

interpretation function $\cdot^{\mathcal{I}}$. For $\sharp(S)$ the cardinality of a set S , \mathcal{I} is such that the following equations hold:

$$\begin{array}{ll}
\top_n^{\mathcal{I}} \subseteq (\Delta^{\mathcal{I}})^n & \epsilon^{\mathcal{I}} = \{\langle \mathbf{x}, \mathbf{x} \rangle \mid \mathbf{x} \in \Delta^{\mathcal{I}}\} \\
\mathbf{p}^{\mathcal{I}} \subseteq \top_n^{\mathcal{I}} & (\mathbf{r}|_{\$i, \$j})^{\mathcal{I}} = \{\langle \mathbf{x}_i, \mathbf{x}_j \rangle \mid \langle \mathbf{x}_1, \dots, \mathbf{x}_n \rangle \in \mathbf{r}^{\mathcal{I}}\} \\
(\$i/n: \mathbf{C})^{\mathcal{I}} = \{\langle \mathbf{d}_1, \dots, \mathbf{d}_n \rangle \in \top_n^{\mathcal{I}} \mid \mathbf{d}_i \in \mathbf{C}^{\mathcal{I}}\} & (\mathbf{e}_1 \circ \mathbf{e}_2)^{\mathcal{I}} = \mathbf{e}_1^{\mathcal{I}} \circ \mathbf{e}_2^{\mathcal{I}} \\
(\neg \mathbf{r})^{\mathcal{I}} = \top_n^{\mathcal{I}} \setminus \mathbf{r}^{\mathcal{I}} & (\mathbf{e}_1 \sqcup \mathbf{e}_2)^{\mathcal{I}} = \mathbf{e}_1^{\mathcal{I}} \cup \mathbf{e}_2^{\mathcal{I}} \\
(\mathbf{r}_1 \sqcap \mathbf{r}_2)^{\mathcal{I}} = \mathbf{r}_1^{\mathcal{I}} \cap \mathbf{r}_2^{\mathcal{I}} & (\mathbf{e}^*)^{\mathcal{I}} = (\mathbf{e}^{\mathcal{I}})^* \\
\top^{\mathcal{I}} = \Delta^{\mathcal{I}} & \mathbf{A}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \quad (\neg \mathbf{C})^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus \mathbf{C}^{\mathcal{I}} \quad (\mathbf{C}_1 \sqcap \mathbf{C}_2)^{\mathcal{I}} = \mathbf{C}_1^{\mathcal{I}} \cap \mathbf{C}_2^{\mathcal{I}} \\
& (\exists \mathbf{e}.\mathbf{C})^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \exists d' \in \mathbf{C}^{\mathcal{I}}. \langle d, d' \rangle \in \mathbf{r}^{\mathcal{I}}\} \\
& (\exists \$i[\mathbf{r}])^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \exists \langle \mathbf{d}_1, \dots, \mathbf{d}_n \rangle \in \mathbf{r}^{\mathcal{I}}. d_i = d\} \\
& (\leq k[\$i]\mathbf{r})^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \sharp(\langle \mathbf{d}_1, \dots, \mathbf{d}_n \rangle \in \mathbf{r}^{\mathcal{I}} \mid d_i = d) \leq k\}
\end{array}$$

A *schema* \mathcal{S} is a set of axioms of the form $\mathbf{C}_1 \sqsubseteq \mathbf{C}_2$ and $\mathbf{r}_1 \sqsubseteq \mathbf{r}_2$. An interpretation \mathcal{I} satisfies $\mathbf{C} \sqsubseteq \mathbf{D}$ ($\mathbf{r} \sqsubseteq \mathbf{s}$) iff $\mathbf{C}^{\mathcal{I}} \subseteq \mathbf{D}^{\mathcal{I}}$ ($\mathbf{r}^{\mathcal{I}} \subseteq \mathbf{s}^{\mathcal{I}}$) and it satisfies \mathcal{S} iff it satisfies all axioms in \mathcal{S} . If \mathcal{I} satisfies a concept, axiom or schema X , we call \mathcal{I} a model of X and write $\mathcal{I} \models X$.

2.2 Syntax and Semantics of Conjunctive Queries

A conjunctive query q has the form $\langle \vec{x} \rangle \leftarrow \text{body}(\vec{x}; \vec{y}; \vec{c})$, such that \vec{x} is a vector of (distinguished) variables, \vec{y} is a vector of (non-distinguished) variables, \vec{c} is a vector of constants from $\Delta^{\mathcal{I}}$, and \vec{x} , \vec{y} , and \vec{c} are mutually disjoint. The query body $\text{body}(\vec{x}; \vec{y}; \vec{c})$ is a conjunction of concept atoms $t: \mathbf{C}$, role atoms $\vec{t}: \mathbf{r}$, and regular expression atoms $\langle t_2, t_1 \rangle: \mathbf{e}$ such that t (possibly indexed) is an element and \vec{t} is a tuple composed of elements from \vec{x} , \vec{y} , or \vec{c} . Without loss of generality we assume that for all expressions of the form $\langle t_i, t_j \rangle: \mathbf{r}|_{\$i, \$j}$, the $\$i, \j are in ascending order. The cardinality of \vec{x} is called the arity of q .

If \mathcal{I} is a model for a schema \mathcal{S} and q a query with arity n , $q^{\mathcal{I}}$ w.r.t \mathcal{S} is a set of n -tuples $\langle \mathbf{o}_1, \dots, \mathbf{o}_n \rangle$ with $\mathbf{o}_i^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ for $1 \leq i \leq n$, such that after replacing each x_i with \mathbf{o}_i , the formula $\exists \vec{y}. \text{body}(\vec{x}; \vec{y}; \vec{c})$ is true in \mathcal{I} . We call a query q with arity 0 a boolean query.

2.3 Query Containment

If q and q' are conjunctive queries for a schema \mathcal{S} with the same arity, q is contained in q' w.r.t. \mathcal{S} , written as $\mathcal{S} \models q \sqsubseteq q'$, iff $q^{\mathcal{I}} \subseteq q'^{\mathcal{I}}$ for every model \mathcal{I} of \mathcal{S} . In other words, q is contained in q' if there is no model for \mathcal{S} and q that is also a model for $\neg q'$, i.e., $\langle \mathcal{S}, q \cup \neg q' \rangle$ is not satisfiable.

Query containment can also be used to answer queries against a schema \mathcal{S} and an ABox \mathcal{A} . To achieve this, the more specific query q functions as ABox, i.e., it contains only constants and no variables, i.e., \vec{x} and \vec{y} are empty. From now on we assume that this is the case, since query answering is the main focus

of this paper. We further assume that q' is a boolean query and contains no distinguished variables, i.e., \vec{x} is empty. The answer for a non-boolean query q' can be computed by using possibly several boolean queries with the distinguished variables replaced with constants in such a way that all possible combinations of replacements are tested.

3 Deciding Query Containment

To decide if a query q is contained in a query q' , we need to form the negation of q' . This is simple if the query contains only concept atoms, because the considered DL provides full concept negation. Role negation is, however, not supported. To overcome this problem the query q' is transformed into a single concept assertion.

3.1 Query Transformation

The transformation is based on a so called *tuple-graph*, which is a graph with labelled nodes. If \mathcal{G} is a tuple-graph for the query q' , \mathcal{G} contains a node, called an individual node, for each element t in \vec{y} and \vec{c} (\vec{x} is by assumption empty) and a node, called a tuple node, for each \vec{t} in q' . An individual node t is labelled with the conjunction of its *name-formula* N_t and all \mathbf{C} such that $t:\mathbf{C}$ appears in q' . A name-formula N_t , also called a representative concept, represents an individual in an interpretation \mathcal{I} after the assertion $t:N_t$ is added to the ABox and N_t is a previously unused concept name. A tuple node \vec{t} is labelled with the conjunction of all \mathbf{r} and \mathbf{e} such that $\vec{t}:\mathbf{r}$ (resp. $\langle t_1, t_2 \rangle:\mathbf{e}$) appears in q' . For a node n , the function $\mathcal{L}(n)$ returns the label of n . For each tuple node $\langle t_1, \dots, t_n \rangle:\mathbf{r}$ ($\langle t_1, t_2 \rangle:\mathbf{e}$) there are edges to the nodes t_i , $1 \leq i \leq n$ (resp. t_1 and t_2). A tuple-graph is composed of one or more connected components, each containing one or more individual nodes, possibly linked to tuple nodes. The graph is used to compute concepts representing each component. For the computation we recursively traverse \mathcal{G} , starting by visiting, for each component, an individual node t_0 in the component, and proceed as follows:

When an individual node t is visited, the node is marked and while there is a connected unmarked tuple node \vec{t} of arity n and t is the i^{th} element in \vec{t} , this node is visited; then $\mathcal{L}(t)$ is set to $\mathcal{L}(t) \sqcap \exists [\mathbf{i}](\mathcal{L}(\vec{t}) \sqcap_{1 \leq j \leq n, j \neq i} (\mathbf{j}/n: \mathcal{L}(t_j)))$.

When a tuple node \vec{t} is visited, the node is marked and while there is a connected unmarked individual node t this node is visited.

After the traversal, the concept expression $\mathcal{L}(t_0)$ represents the component of t_0 . All N_t that represent a non-distinguished variable t that does not occur in a cycle in \mathcal{G} can be replaced with \top in $\mathcal{L}(t_0)$. After the traversal of all components, we build an ABox \mathcal{A} containing all atoms in q and for each starting node t_0 of

a component, we set \mathcal{A} to $\mathcal{A} \cup \{t_0: \neg\mathcal{L}(t_0)\}$ if t_0 represents a constant or to $\mathcal{A} \cup \{\top \sqsubseteq \neg\mathcal{L}(t_0)\}$ if t_0 represents a non-distinguished variable. The query q is contained in the query q' w.r.t. \mathcal{S} iff $\langle \mathcal{S}, \mathcal{A} \rangle$ is unsatisfiable.

If the logic allows n-ary relations, as is the case for \mathcal{DLR} , a further translation can be used in order to be able to use existing reasoning algorithms. Calvanese et al. [3] translate \mathcal{DLR}_{reg} to the modal logic *converse-PDL_g*. The n-ary relations are replaced with name-formulae and a technique called reification is used to identify the name-formulae with the original relation in a model. Horrocks et al. [10] use the Description Logic *SHIQ* and translate a \mathcal{DLR} query to a *SHIQ* ABox using the same technique to eliminate the n-ary relations. We do not perform a further translation here, since the problems that arise from a cyclic query are independent of the chosen translation.

3.2 Cyclic Queries

When the tuple-graph \mathcal{G} contains a cycle including nodes that represent non-distinguished variables, the name-formulae for these variables cannot simply be replaced by the concept \top without losing the coreference that closes the cycle. In [3] it is suggested to replace the non-distinguished variables in a cycle with constants from the more specific query q , since cycles cannot directly be expressed in a \mathcal{DLR} schema. Cycles can, however, sometimes be eliminated by identifying two variables with the same element, as suggested in [10]. Therefore, additional non-determinism is introduced by equating every possible combinations of non-distinguished variables in a cycle with each other. Another source for cycles arises from the reflexive transitive closure expressions, as the following example highlights. This case is not yet handled properly by the existing approaches.

3.3 An Incompleteness Example

For the sake of simplicity, we assume that the schema \mathcal{S} is empty. Let \mathbf{a} be a constant, \mathbf{x} , \mathbf{y} , and \mathbf{z} non-distinguished variables, q the query $\langle \rangle \leftarrow \mathbf{a}: \exists \mathbf{r}|_{\mathcal{S}_1, \mathcal{S}_2}. \exists \mathbf{s}|_{\mathcal{S}_1, \mathcal{S}_2}. \exists \mathbf{s}|_{\mathcal{S}_1, \mathcal{S}_2}. \top$ and q' the query $\langle \rangle \leftarrow \langle \mathbf{x}, \mathbf{y} \rangle: \mathbf{s} \wedge \langle \mathbf{y}, \mathbf{z} \rangle: \mathbf{s} \wedge \langle \mathbf{x}, \mathbf{z} \rangle: (\mathbf{s}|_{\mathcal{S}_1, \mathcal{S}_2})^*$. We now want to determine if q is contained in q' w.r.t. \mathcal{S} , i.e., if $\mathcal{S} \models q \sqsubseteq q'$. Since q contains only constants and no variables, we could also regard this as answering the query q' against the knowledge base $\langle \mathcal{S}, q \rangle$. To answer the query we compute the concept that represents the tuple-graph \mathcal{G} (see Fig. 1) for q' (in this case \mathcal{G} has only one component). We randomly chose the node \mathbf{x} as t_0 and the first tuple node visited is $\langle \mathbf{x}, \mathbf{y} \rangle: \mathbf{s}$. The edge labels in Fig. 1 indicate the order in which the nodes were visited. The first label that is modified is the label for the individual node \mathbf{z} and it is extended to $\mathbf{N}_z \sqcap \exists [\mathbf{2}]((\mathbf{s}|_{\mathcal{S}_1, \mathcal{S}_2})^* \sqcap \mathbf{1}/2: \mathbf{N}_x)$. Then the label for \mathbf{y} is changed to $\mathbf{N}_y \sqcap \exists [\mathbf{1}](\mathbf{s} \sqcap \mathbf{2}/2: (\mathbf{N}_z \sqcap \exists [\mathbf{2}]((\mathbf{s}|_{\mathcal{S}_1, \mathcal{S}_2})^* \sqcap \mathbf{1}/2: \mathbf{N}_x)))$ and finally

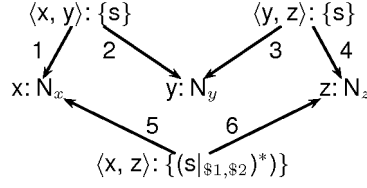


Figure 1: The tuple-graph for q' .

the label of \mathbf{x} becomes $\mathbf{N}_x \sqcap \exists [\$1](\mathbf{s} \sqcap \$2/2: (\mathbf{N}_y \sqcap \exists [\$1](\mathbf{s} \sqcap \$2/2: (\mathbf{N}_z \sqcap \exists [\$2](\mathbf{s}|_{\$1, \$2})^* \sqcap \$1/2: \mathbf{N}_x))))$.

Since the tuple graph for q' is cyclic, we cannot simply replace the name-formulae for the non-distinguished variables with \top . Instead, we replace the non-distinguished variables (and their corresponding name-formulae) with constants or variables from q , i.e., \mathbf{a} (resp. \mathbf{N}_a) in this example. It is easy to see that the replacement does not lead to a correct solution. The resulting knowledge base $\langle \mathcal{S}, q \cup \{\mathbf{a}: \neg(\mathcal{L}(t_0)_{[N_x \leftarrow N_a, N_y \leftarrow N_a, N_z \leftarrow N_a]})\} \rangle$, where $X_{[a \leftarrow b]}$ denotes the replacement of each occurrence of a in X with b , is satisfiable and therefore q is not contained in q' resp. the knowledge base $\langle \mathcal{S}, q \rangle$ does not entail q' . This is, however, incorrect, since the transitive reflexive closure $\langle \mathbf{x}, \mathbf{z} \rangle: (\mathbf{s}|_{s1, s2})^*$ automatically holds if the two tuples $\langle \mathbf{x}, \mathbf{y} \rangle: \mathbf{s} \wedge \langle \mathbf{y}, \mathbf{z} \rangle: \mathbf{s}$ hold. If q'' is the acyclic query $\langle \rangle \leftarrow \langle \mathbf{x}, \mathbf{y} \rangle: \mathbf{s} \wedge \langle \mathbf{y}, \mathbf{z} \rangle: \mathbf{s}$, the query q is clearly contained in q'' , since a model for q is always also a model for q'' , and therefore q should also be contained in q' .

Since different variables may be bound to the same element, Horrocks et al. suggested to non-deterministically equate all possible combinations of non-distinguished variables. In some cases this eliminates the cycle and allows to determine a solution for the resulting acyclic query. For example, the query $\langle \rangle \leftarrow \langle \mathbf{x}, \mathbf{y}_1 \rangle: \mathbf{r} \wedge \langle \mathbf{x}, \mathbf{y}_2 \rangle: \mathbf{r} \wedge \langle \mathbf{y}_1, \mathbf{z} \rangle: \mathbf{s} \wedge \langle \mathbf{y}_2, \mathbf{z} \rangle: \mathbf{s}$ can be transformed to an acyclic query if \mathbf{y}_2 is treated as equal to \mathbf{y}_1 . The resulting equi-satisfiable query $\langle \rangle \leftarrow \langle \mathbf{x}, \mathbf{y}_1 \rangle: \mathbf{r} \wedge \langle \mathbf{y}_1, \mathbf{z} \rangle: \mathbf{s}$ is acyclic and the non-distinguished variables in the concept constructed during the traversal of the tuple-graph can simply be replaced with \top . In the example given before this does not work, since all resulting queries from equating non-distinguished variables are still cyclic.

In Horrocks et al. the queries are not allowed to contain regular expressions, which makes it impossible to express the problematic query given above. However, since the employed underlying logic *SHIQ* supports transitive roles, the same problem arises when the role \mathbf{s} is transitive and the query q' is modified to $\langle \rangle \leftarrow \langle \mathbf{x}, \mathbf{y} \rangle: \mathbf{s} \wedge \langle \mathbf{y}, \mathbf{z} \rangle: \mathbf{s} \wedge \langle \mathbf{x}, \mathbf{z} \rangle: \mathbf{s}$.

4 Eliminating Transitivity Cycles

As a solution for the problem described above, we suggest to eliminate cycles caused by transitive roles in the query q' . Let \mathcal{G} be the tuple-graph for q' . If \mathcal{G} contains a path $(t_0, \vec{t}_0, t_1, \vec{t}_1, \dots, t_{n-1}, \vec{t}_{n-1}, t_n)$, $n > 1$ and a path (t_0, \vec{t}, t_n) and $\mathcal{L}(\vec{t})$ contains $(\mathbf{s}|_{\$k, \$l})^*$ and all $\mathcal{L}(\vec{t}_i)$, $0 \leq i < n$, contain an expression $\mathbf{s}'|_{\$k, \$l}$ such that $\mathbf{s}' \sqsubseteq^* \mathbf{s}$, then we produce \mathcal{G}' from \mathcal{G} by removing $(\mathbf{s}|_{\$k, \$l})^*$ from $\mathcal{L}(\vec{t})$. If \mathcal{G}' contains a tuple node $\langle t_1, t_2 \rangle$ with an empty label, we remove $\langle t_1, t_2 \rangle$ and all outgoing edges. We claim that the concept in the label of t_0 after the traversal of \mathcal{G}' is equi-satisfiable with the concept resulting from the traversal of \mathcal{G} . This is due to the fact that the deleted labels (resp. nodes and edges) are only shortcuts, and are necessarily contained in every model of the knowledge base. If the query language contains no regular expressions, but the DL, e.g., *SHIQ*, supports transitive roles, a similar shortcut elimination is possible.

After the elimination of the shortcuts all cycles in the tuple-graph must be caused by individuals that are explicitly present in the ABox (resp. in the more specific query), hence the existing algorithm can be applied.

5 Conclusion

Query answering (resp. query containment) algorithms are difficult to develop, and in particular cycles in a query can cause problems. We highlighted one such problem by means of an example that is not answered correctly with the existing algorithms. The problems are mainly due to transitivity, but also the extension suggested in [12, 10] to equate non-distinguished variables in a cycle has to be taken into account.

As a solution we suggested eliminating cycles that are caused by transitivity by deleting some atoms from the query. This is possible, since all the deleted atoms are necessarily satisfied by every model of the knowledge base. The resulting query contains only cycles that are caused by individuals that are explicitly named in the knowledge base and therefore the existing algorithms can be used. As a part of our future work, we will formally prove the correctness of an extended algorithm.

We did not yet take into account an underlying logic that supports nominals, e.g., the DL *SHOIQ*. In the presence of nominals, even without transitivity, cycles are not necessarily caused by individuals explicitly present in the knowledge base. Therefore, the suggested shortcut elimination is not enough to answer queries that include a cycle with non-distinguished variables. We will try to develop an algorithms for query answering for a DL that supports nominals in our future work.

References

- [1] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. Patel-Schneider, and L. A. Stein. OWL web ontology language reference. Technical report, W3C, 2004. <http://www.w3.org/TR/2004/REC-owl-ref-20040210/>.
- [2] T. Berners-Lee, M. Fischetti, and M. L. Dertouzos. *Weaving the Web*. Harper San Francisco, 1999.
- [3] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. ACM Press, 1998.
- [4] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Description logic framework for information integration. In *Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning*, 1998.
- [5] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. of the 9th Annual ACM Symposium on Theory of Computing*. ACM Press, 1977.
- [6] V. Haarslev, R. Möller, and M. Wessel. RACER user's guide and reference manual, version 1.7.19. URL, <http://www.sts.tu-harburg.de/~r.f.moeller/racer/racer-manual-1-7-19.pdf>, 2004.
- [7] I. Horrocks. FaCT and iFaCT. In *Proc. of the Int. Workshop on Description Logics*, 1999.
- [8] I. Horrocks and P. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. *Journal of Web Semantics*, (4), 2004.
- [9] I. Horrocks and U. Sattler. A tableaux decision procedure for SHOIQ. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence*, 2005. To appear.
- [10] I. Horrocks, U. Sattler, S. Tessaris, and S. Tobies. How to decide query containment under constraints using a description logic. In *Proc. of the 7th Int. Conf. on Logic for Programming and Automated Reasoning*. Springer-Verlag, 2000.
- [11] Pellet. Pellet OWL reasoner, 2003. <http://www.mindswap.org/2003/pellet/>.
- [12] Sergio Tessaris. *Questions and answers: reasoning and querying in Description Logic*. Phd thesis, University of Manchester, 2001.