Scalable Instance Retrieval for the Semantic Web by Approximation

Holger Wache[1], Perry Groot[2], and Heiner Stuckenschmidt[1]

[1] Vrije Universiteit Amsterdam, de Boelelaan 1081a,
1081HV Amsterdam, The Netherlands
{holger, heiner}@cs.vu.nl
[2] Radboud University Nijmegen, Toernooiveld 1,
6500GL Nijmegen, The Netherlands
Perry.Groot@science.ru.nl

Abstract. Approximation has been identified as a potential way of reducing the complexity of logical reasoning. Here we explore approximation for speeding up instance retrieval in a Semantic Web context. For OWL ontologies, i.e., Description Logic (DL) Knowledge Bases, it is known that reasoning is a hard problem. Especially in instance retrieval when the number of instances that need to be retrieved becomes very large. We discuss two approximation methods for retrieving instances to conjunctive queries over DL T-Boxes and the results of experiments carried out with a modified version of the Instance Store System.

1 Motivation

A central issue in the Semantic Web research community is the expressivity of its underlying language and the complexity of the reasoning services it supports. There is a direct correspondence between the current Semantic Web ontology language OWL and Description Logic (DL).[1] Research in DL has lead to sophisticated DL reasoners [6, 3, 5] that can be used to reason with OWL ontologies on the Semantic Web. Considering T-Box reasoning, current state of the art techniques seem capable of dealing with real world ontologies [7, 4]. However, besides T-Box reasoning, an important application domain of ontologies is A-Box reasoning, i.e., reasoning and retrieving the individuals in an ontology. Experiments have shown that state of the art DL reasoners break down for A-Box reasoning when the number of instances becomes large [8]. Present work focusses at approximation techniques to make A-Box reasoning in DLs more scalable when retrieving instances from an ontology with a large number of instances.

In this paper, we investigate optimization techniques that are based on approximate logical reasoning. The underlying idea of these techniques is to replace certain inference problems by simpler problems such that either the soundness or the completeness, but not both, of the solutions is preserved. The solutions to the simpler problems are approximate solutions to the original problem.

[1] More precisely two of the three species of OWL.

M. Dean et al. (Eds.): WISE 2005 Workshops, LNCS 3807, pp. 245–254, 2005.

The contribution of this work is in comparing the performance of two approximate reasoning methods proposed in the literature applied to the real world task of answering conjunctive queries over DL Knowledge Bases. For this, we used the Instance Store [8], a state of the art system developed to scale-up instance retrieval for ontologies with a large number of instances, and extended it with two approximation techniques. The Gene Ontology is used as benchmark data set to evaluate the performance of the approximation techniques.

The paper is organized as follows. Section 2 gives a brief introduction to DLs. Section 3 defines the problem of instance retrieval in the context of Description Logics, which is restricted to conjunctive queries. Section 4 gives a brief overview of two approximation methods and describes how they can be applied to the problem of instance retrieval. Section 5 gives the results of experiments with the two approximation methods applied to instance retrieval using the Gene Ontology. Section 6 concludes our work.

2 Description Logics

DLs [1] are a special type of logic that is tailored to define terminological knowledge in terms of sets of objects with common properties. Recently, DLs have become popular as a formal foundation for the Web Ontology Language OWL. The basic modelling elements of a DL are instances, concepts, and relations. These modelling elements are provided with a formal semantics in terms of an abstract domain interpretation $\mathcal{I}$, which maps each instance onto an element of an abstract domain Δ. Instances can be connected by binary relations defined as subsets of $\Delta \times \Delta$. Concepts are interpreted as a subset of the abstract domain Δ. Intuitively, a concept is a set of instances that share certain properties. These properties are defined in terms of concept expressions. Typical operators are the Boolean operators as well as universal and existential quantification over relations to instances in other concepts. The formal definitions can be found in the first table.

DL Expr.	Semantics
A	$A^{\mathcal{I}} \subseteq \Delta$
$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta - C^{\mathcal{I}}$
$C \sqcap D$	$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$C \sqcup D$	$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
$\exists R.C$	$(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y : (x,y) \in R^{\mathcal{I}}\}$
$\forall R.C$	$(\forall R.C)^{\mathcal{I}} = \{x \mid (x,y) \in R \implies y \in C^{\mathcal{I}}\}$

DL Axiom	Semantics
$C(x)$	$x^{\mathcal{I}} \in C^{\mathcal{I}}$
$P(x,y)$	$(x^{\mathcal{I}}, y^{\mathcal{I}}) \in P^{\mathcal{I}}$
$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
$P \sqsubseteq R$	$P^{\mathcal{I}} \subseteq R^{\mathcal{I}}$
$P \equiv R^-$	$P^{\mathcal{I}} = \{(x,y) \mid (y,x) \in R^{\mathcal{I}}\}$

A DL Knowledge base consists of a set of axioms about instances, concepts (potentially defined in terms of complex concept expressions), and relations. Axioms can be used to state that an instance belongs to a concept and that two instances are in a certain relation. Other type of axioms describe relations between concepts and instances. It can be stated that one concept is a

subconcept of the other (all its instances are also instances of this other concept). Further, we can define a relation to be a subrelation or the inverse of another relation. The formal definition of axioms can be found in the second table.

The formal semantics of concepts and relations as defined by the interpretation into Δ can be used to automatically infer new axioms from existing definitions. In particular, given an ontology and a number of instance related axioms, we can automatically determine whether an instance belongs to a certain concept based on the expression defining the concept.

3 Instance Retrieval Queries

In this article we focus on the following instance retrieval problem:

Definition 1 (Instance retrieval w.r.t. some query). *Given an A-Box $\mathcal{A}$ and a query Q, i.e., a concept expression, find all individuals a such that a is an instance of Q, i.e., $\{a \mid \forall a \in \mathcal{A}, a : Q\}$.*

Often, an analogy is made between databases (DBs) and DL KBs. The schema of a DB corresponds to the T-Box and the DB instances correspond to the A-Box. However, A-Boxes have a very different semantics. This makes query answering in a DL setting often much more complex than query answering in a DB. Given the expressivity of DLs, retrieving instances to a query cannot simply be reduced to model checking as in the database framework because there is no single minimal model for a query. Knowledge Bases may contain nondeterminism and/or incompleteness. Therefore, deductive reasoning is needed when answering a query in a DL setting.

Conjunctive Queries. A-Box query languages have been quite weak for earlier DL systems. Usually they supported very simple A-Box queries like instantiation (is individual i an instance of concept C, i.e., $i : C$), realisation (what are the most specific concepts i is an instance of), and retrieval (which individuals are instances of concept C).

In [9] an approach for answering conjunctive queries over arbitrary DL KBs is given based on the translation of the query into an equivalent concept expression, i.e., by *rolling up* the query.

Definition 2 (Boolean Conjunctive Query). *A* Boolean conjunctive query *Q is of the form $q_1 \wedge \cdots \wedge q_n$, where $q_1, \ldots, q_n$ are query 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 either individual names or variables.*

Because binary relations in a conjunctive query can be translated into an existential restriction such that logical consequence is preserved, standard DL inference methods can then be used to classify the concept expression the query is translated into as well as retrieve the instances that belong to it. [9] enables us to use an expressive query language for arbitrary expressive DL KBs.

Instance Store. DL reasoning is hard, especially in the case of instance retrieval when the number of instances grows very large. To speed up the overall cost of instance retrieval, one can address the number and cost of checking whether a single instance belongs to a query.

Instance Store [8] is developed to speed up instance retrieval by replacing costly instantiation checks $a : Q$ with database retrieval. However, Instance Store can not replace all DL reasoning steps using database retrieval. In some situations DL instantiation checks must still be performed. An analysis of the Instance Store revealed a drastic breakdown in performance in these situations, which hampers its goal to scale-up reasoning to ontologies with a large number of instances. At the moment Instance Store only supports role-free A-Boxes, i.e., relationships between instances in the A-Box are not allowed, but this was sufficient for our purpose.

4 Approximation Techniques for Instance Retrieval

There are three components of the instance retrieval problem where approximation methods can be applied:

The Query. The query can be made weaker, i.e., more general, by omitting or replacing parts of the query. The underlying assumption is that simpler queries are easier to check.

The Ontology. We assume that the query is formulated relative to a given ontology. Concept expressions in the ontology (representing for example an instantiation check) can be approximated by weaker or stronger concept expressions.

The Instance Descriptions. In order to check whether instances belong to the query, first the descriptions of instances are translated into equivalent concept expressions. Consequently, those concept expressions can be approximated by weaker or stronger concept expressions.

This section reviews the techniques of [10] and [11] that can be used to approximate instance retrieval in DL. Figure 1 gives an overview of the various components used in instance retrieval. The method of [10] was proposed to approximate satisfiability of concept expressions (usable in step 5 of Figure 1).[2] The method of [11] can be used to approximate conjunctive queries, or its concept expression counterpart (usable in steps 1 and 2 of Figure 1).

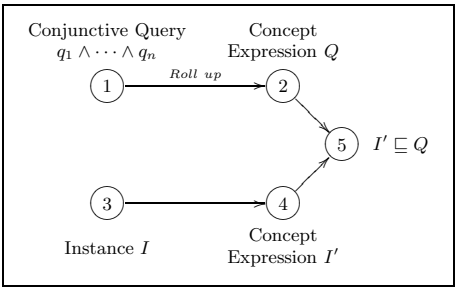

Fig. 1. Various components

[2] [10] should also be usable in steps 2 and 4, although not proposed originally.

Both methods propose to approximate an instantiation test using a sequence of tests $C_1, \ldots, C_n$. Assuming that less complex tests can be answered in less time, instance checking can then be speeded up. However, both methods differ in their strategy for selecting the sequence of expressions C_i to be checked successively. In general, [11] argues that the order should balance two factors:

1. The *smoothness* of the approximation. In particular, the next test C_{i+1} should lead to the next best approximation.
2. The potential contribution of the extension of C_{i+1} to the *time complexity* of the tests to be done by the system.

Approximating DL Satisfiability. In DLs, satisfiability checking can be seen as the most basic task as many reasoning services can be restated into satisfiability checks [1]. In [10] a technique has been developed to approximate satisfiability checks. Concept expressions are approximated by two sequences $C_1, \ldots, C_n$ of simpler concept expressions, obtained by syntactic manipulations, which can be used to determine the satisfiability of the original concept expression.

For every subconcept D, [10] defines the *depth* of D to be 'the number of universal quantifiers occurring in C and having D in its scope'. The scope of $\forall R.\phi$ is ϕ which can be any concept term containing D. A sequence of weaker (stronger) approximated concepts can be defined, denoted by $C_i^\top$ ($C_i^\bot$), by replacing every *existentially quantified* subconcept, i.e., $\exists R.\phi$ where ϕ is any concept term, of depth greater or equal than i by $\top$ ($\bot$). Concept expressions are assumed to be in negated normal form (NNF) before approximating them.

Theorem 1 ([10]). *For each i, if $C_i^\top$ is unsatisfiable then $C_j^\top$ is unsatisfiable for all $j \geq i$, hence C is unsatisfiable. For each i, if $C_i^\bot$ is satisfiable then $C_j^\bot$ is satisfiable for all $j \geq i$, hence C is satisfiable.*

The sequences $C^\top$ and $C^\bot$ can be used to gradually approximate the satisfiability of a concept expression. [10] only replaces subconcepts $D \equiv \exists R.C$ as the worst case complexity depends on the nesting of existential and universal quantifiers. Theorem 1 leads to the following for $C^\bot$-approximation:

$$\begin{array}{llll} (I \sqsubseteq Q)_i^\bot & \text{is not satisfiable} & \Leftrightarrow (I \sqcap \neg Q)_i^\bot & \text{is satisfiable} \quad \Rightarrow \\ (I \sqcap \neg Q) & \text{is satisfiable} & \Leftrightarrow (I \sqsubseteq Q) & \text{is not satisfiable} \end{array}$$

Therefore, we are only able to reduce complexity when approximated subsumption tests are not satisfiable. When an approximated subsumption test $(I \sqsubseteq Q)_i^\bot$ is satisfiable, nothing can be concluded and the approximation continues to level $i+1$ until no more approximation is applicable, i.e., the original concept term is obtained. Analogously, from Theorem 1 one obtains that when $(I \sqsubseteq Q)_i^\top$ is satisfiable this implies that $(I \sqsubseteq Q)$ is satisfiable. When $(I \sqsubseteq Q)_i^\top$ is not satisfiable nothing can be deduced and the approximation continues to level $i+1$.

Research on this kind of DL approximation is quite limited. [10] is the only method that deals with approximation of satisfiability in DLs. Few results have only been obtained recently [2].

Approximating Conjunctive Queries. In [11] a method is introduced for approximating conjunctive queries. The method computes a sequence $Q^1, \ldots, Q^n$ of queries such that: (1) $i < j \Rightarrow Q^i \sqsupseteq Q^j$ and (2) $Q^n \equiv Q$. The first property ensures that the quality of the results of the queries doesn't decrease. The second property ensures that the last query computed returns the desired exact result.

The proposed method can easily be adapted for instantiation checks. The computed sequence $Q^1, \ldots, Q^n$ is used to generate the sequence $C_1^\Delta, \ldots, C_n^\Delta$ with $C_i^\Delta = a : Q^i$. Assuming that less complex queries can be answered in less time, instantiation checks can then be speeded up using the following implication:

$$(I \not\sqsubseteq Q') \wedge (Q \sqsubseteq Q') \Rightarrow I \not\sqsubseteq Q$$

In [11] the sequence of subsuming queries $Q^1, \ldots, Q^n$ is constructed by stepwise adding a conjunct (of the original query) starting with the universal query.

A problem that remains to be solved in this approach is a strategy for selecting the sequence of queries to be checked successively. This problem boils down to ordering the conjuncts of the query which should balance the two factors 'smoothness' and 'time complexity'.

As described in [11] the smoothness of the approximation can be guaranteed by analyzing the dependencies between variables in the query. After translating the conjunctive query to a DL expression, these dependencies are reflected in the nesting of subexpressions. As the removal of conjuncts from a concept expression is equivalent to substitution by $\top$, this nesting provides us with a selection strategy to determine a sequence of approximations S_i where all subexpressions at depth greater or equal than i are replaced by $\top$. Hence, this method is somewhat similar to $C^\top$-approximation except that it is restricted to the conjunctive query, i.e., the instance description is not approximated, and it can replace any conjunct in the query with $\top$, not only existentially quantified conjucts.

Typically, however, queries often have a very flat structure. For example, all queries used in our experiments with the Gene Ontology are of depth one. This means that S_0 is the query $\top$ whereas S_1 is already the original query. To avoid this bad approximation scheme, next we propose an improved strategy.

An Improved Approximation Strategy. To overcome the flatness of queries typically found in ontologies, we propose a strategy that also provides an order for subexpressions at the same level of depth. A possible ordering is the expected time contribution of a conjunct to the costs of the subsumption test. As measuring the actual time is practically infeasible, a heuristic is proposed.

For this purpose, we unfold the conjuncts using the definitions of the concepts from the ontology occurring in the conjunct. In order to determine a suitable measure of complexity for expressions, we consider the standard proof procedure for DLs. Most existing DL reasoners are based on tableau methods, which determine the satisfiability of a concept expression by constructing a constraint system based on the structure of the expression. As the costs of checking the satisfiability of an expression depends on the size of the constraint system, we can use this size as a measure of complexity. As determining the exact size of

the constraint system requires to run the tableau method, heuristics are used for estimating the size. Based on this estimated size, we determine the order in which conjuncts at the same level of depths are considered.

In the following, we propose a method for estimating the size of the tableau for expressions in $\mathcal{ALC}$ that will be used in the experiments. The tableau rules [1] provide us with quite a good idea about an estimation of the maximal size of the tableau in the worst case. For this purpose, we define a function Φ that assigns a natural number representing the estimated size of the corresponding constraint system to an arbitrary $\mathcal{ALC}$ expression in the following way:

$$\begin{aligned}
\Phi(A) &= 1 \\
\Phi(\neg A) &= 0 \\
\Phi(C \sqcap D) &= 2 + \Phi(C) + \Phi(D) \\
\Phi(C \sqcup D) &= \phi + 2 + \Phi(C) + \Phi(D) \text{ where } \phi \text{ is the current value of } \Phi(E) \\
\Phi(\exists\, R.C) &= 2 + \Phi(C) \\
\Phi(\forall\, R.C) &= n + n \cdot \Phi(C) \text{ where } n \text{ is the number of existential quantifiers in } E
\end{aligned}$$

A and $\neg A$: Atomic concepts are added as a single constraint. Negated concepts are not added as they are merely used to check the existence of a contradiction.

$C \sqcap D$: Two new constraints are added. The expressions in these constraints have to be evaluated recursively, therefore, we also have to estimate the number of constraints that will be generated by C and D.

$C \sqcup D$: Two new constraints are added and each of the constraints has to be evaluated recursively, however, we have to deal with two separate constraint systems from this point on. The number of constraints in the system at this point has to be doubled. For an estimation we add the current estimation value.

($\exists\, R.C$): Two new constraints are added, one for the relation and one restricting the object in the relation to C Object y has to be evaluated recursively.

($\forall\, R.C$): A new constraint has to be added for every existing constraint xRy in the constraint system S and each one has to be evaluated recursively. As we do not know how many of these statements are or will be in S, we use the overall number of existential quantifiers in the expression that can lead to the addition of these constraints as an upper bound.

The value Φ can now be computed for each conjunct in the query and be used as a basis for determining the order in which conjuncts at the same level of nesting are processed.

5 Experimental Evaluation

In this section experimental results are shown of the approaches described in the previous section. The main question focused on in the experiments is *if*, and if yes, in *what way* does approximation reduce the complexity of the retrieval task. We focus on the number of operations needed and the overall computation time used. The goal of our approximation approach is to replace costly reasoning operations by a (small) number of cheaper approximate reasoning operations. The approximation methods used are sound and complete. Therefore, the suitability

of the approximation methods depend solely on the time gained (or lost) when classical operations are replaced by a number of approximate ones.

Our experiments were made with the Gene ontology and Instance Store [8]. The focus of our experiments are those queries where Instance Store cannot replace all DL reasoning with database retrieval, but must still check the instantiations of some instances. These instantiation checks were found to be a bottleneck in the scalability of this approach. We originally started with 17 queries (with $Q1$ to $Q6$ user formulated queries and queries $Q7$ to $Q17$ artificial), but discarded the queries that didn't require instantiation checks from further experiments.

Table 1. Performed Subsumption tests

	normal			$C^\top$			$C^\perp$			C^Δ		
		true	false		true	false		true	false		true	false
Q2	normal	9	11	$L0$	0	19	$L0$	19	0	$L0$	20	0
				normal	9	11	normal	9	11	$L1$	20	0
										$L2$	9	11
										normal	9	0
Q8	normal	10	597	$L0$	0	606	$L0$	606	0	$L0$	607	0
				normal	10	597	normal	10	597	$L1$	10	597
										normal	10	0
Q12	normal	15	7856	$L0$	0	7871	$L0$	7871	0	$L0$	15	7856
				normal	15	7856	normal	15	7856	normal	15	0
Q14	normal	5	403	$L0$	0	407	$L0$	407	0	$L0$	408	0
				normal	5	403	normal	5	403	$L1$	5	403
										$L2$	5	0
										normal	5	0
Q15	normal	46	6647	$L0$	0	6693	$L0$	6693	0	$L0$	6693	0
				normal	46	6647	normal	46	6647	normal	46	6647
Q17	normal	1	7872	$L0$	0	7873	$L0$	7873	0	$L0$	1	7872
				normal	1	7872	normal	1	7872	normal	1	0

The results of the first experiments are shown in Table 1, which is divided into four columns with each column reporting the number of subsumption tests performed. The first column reports results for the experiment without any approximation, the second column with $C^\top$-approximation, the third column with $C^\perp$-approximation, and the fourth column with C^Δ-approximation. Each column is further divided into smaller rows and columns. The rows represent the level of the approximation used, where *normal* denotes without approximation, and Li denotes the level of the approximation approach. The subcolumns show the number of subsumption tests that resulted in true or false.[3] This distinc-

[3] We will use the shorthand 'true subsumption test' and 'false subsumption test' to indicate these two distinct results.

tion is important, because Section 4 tells us that only when a $C^{\top}$-approximated subsumption succeeds, or a $C^{\bot}$- or C^{Δ}-approximated subsumption test fails we obtain a reduction in complexity.

Discussion. Let us first focus on the question *if* the approximation methods can lead to any reduction in complexity. Table 1 shows that $C^{\top}$- and $C^{\bot}$-approximation cannot reduce the number of normal subsumption tests. Only C^{Δ} is able to reduce, except for $Q15$, all false subsumption tests to 0.

The first column in Table 1 shows that much more false subsumption tests are needed than true subsumption tests. This indicates that $C^{\top}$-approximation is wrong in this approach as it can only be used to lower the complexity of true subsumption tests, which is negligible when compared to false subsumption tests. This may explain its bad approximating behaviour, however, $C^{\bot}$ also performs badly, which does approximate false subsumption tests. Closer analysis shows that *term collapsing* [2], i.e., the substitution of terms by $\top$ or $\bot$ results in the query becoming equivalent to $\top$ or $\bot$, is the reason for this. An analysis of $C^{\bot}$ shows that this occurs in *all cases*.

Apart from looking at *if* an approximation method can successfully reduce the number of normal subsumption tests, we must also consider the cost for obtaining the reduction, i.e., in *what way* are the normal subsumption tests reduced. For example, approximating $Q8$ changes $607 = 10 + 597$ normal subsumption tests into 10 normal subsumption tests, 607 C_1^{Δ} subsumption tests, and 607 C_0^{Δ} subsumption tests. Thus, the number of subsumption tests may increase, but the complexity of most tests will be lower than normal. Note however, that some computations seem unnecessary as nothing can be deduced from them, e.g., the 607 C_0^{Δ} tests.Obviously, in this approach unnecessary subsumption tests should be minimized. Several cases can be observed in the experiments with C^{Δ}-approximation. Either no subsumption test is unnecessary ($Q12, Q17$), some subsumption tests are unnecessary ($Q2, Q8, Q14$), or all subsumption tests are unnecessary ($Q15$).

This distinction seems to influence the overall time needed when approximating a query. Table 2 reports the overall time in milliseconds needed for each query. For comparison $C^{\top}$ and $C^{\bot}$ are also reported. For queries having unnecessary subsumption tests, approximation always leads to more computation time. In those cases, reducing the complexity of subsumption tests do not weigh up to the costs of additional (unnecessary) subsumption tests. For queries having no unnecessary subsumption tests, approximation does save time when compared to the normal case.

Table 2. Time needed for Subsumption tests (in milliseconds)

	normal	$C^{\top}$	$C^{\bot}$	C^{Δ}
Q2	175	348	299	547
Q8	5373	8383	7753	9912
Q12	61410	93100	85764	56478
Q14	4372	6837	6017	7391
Q15	61560	90847	83714	114162
Q17	113289	158218	144689	93074

Another observation of Table 1 is that false subsumption tests for C^{Δ} only occur at *one level*. It seems that the conjunct that is added to the approximated conjunctive query on which the false subsumption tests occur is crucial in determining the outcome.The role of conjunct in a subsumption test is still unclear. More research is needed if this conjunct (or a group of conjuncts) can be identified in advance to speed up approximation.

6 Conclusions

Instance retrieval is one of the most important inferences in the Semantic Web. In order to make methods more scalable for ontologies with a large set of instances we investigated two approximation methods and evaluated them on a benchmark set. Both methods use a similar idea, i.e., removing parts of an expression to make it simpler to speed up retrieval. However, the method of [10] shows bad approximating behaviour because the selection and substitution of subconcepts is too restrictive. The method of [11] was extended with a heuristic for subconcept selection and shows some potential for speeding up instance retrieval. However, more research is needed to improve the heuristic and to determine if the approximation method can be used to speed up instance retrieval.

References

1. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P F. Patel-Schneider. *The Description Logic Handbook - Theory, Implementation and Applications*. Cambridge University Press, 2003.
2. P. Groot, H. Stuckenschmidt, and H. Wache. Approximating Description Logic Classification for Semantice Web Reasoning. In A. Gómez-Pérez and J. Euzenat, editors, *ESWC'2005*, pages 318–332. Springer-Verlag, 2005.
3. V. Haarslev and R. Möller. RACE system description. In *Proceedings of the 1999 DL Workshop*, CEUR Electronic Workshop Proceedings, pages 130–132, 1999.
4. V. Haarslev and R. Möller. High performance reasoning with very large knowledge bases: A practical case study. In *IJCAI'2001*, pages 161–168, 2001.
5. V. Haarslev and R. Möller. RACER system description. In *IJCAR'2001*, volume 2083 of *LNAI*, pages 701–705. Springer, 2001.
6. I. Horrocks. The FaCT System. In *TABLEAUX'98*, volume 1397 of *LNAI*, pages 307–312. Springer, 1998.
7. I. Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In *KR'98*, pages 636–647. Morgan Kaufmann, 1998.
8. I. Horrocks, L. Li, D. Turi, and S. Bechhofer. The Instance Store: DL Reasoning with Large Numbers of Individuals. In *Proc. of the 2004 DL Workshop*, 2004.
9. I. Horrocks and S. Tessaris. A Conjunctive Query Language for Description Logic Aboxes. In *AAAI*, pages 399–404, 2000.
10. M Schaerf and M Cadoli. Tractable reasoning via approximation. *Artificial Intelligence*, 74:249–310, 1995.
11. H. Stuckenschmidt and F. van Harmelen. Approximating terminological queries. In *FQAS'2002*, number 2522 in LNCS, pages 329–343. Springer-Verlag, 2002.