Optimization Techniques for Retrieving Resources Described in OWL/RDF Documents: First Results

Volker Haarslev[†] **and Ralf Möller**[‡]
[†]Concordia University, Montreal
[‡]Technical University Hamburg-Harburg

Abstract

Practical description logic systems play an ever-growing role for knowledge representation and reasoning research even in distributed environments. In particular, the often-discussed semantic web initiative is based on description logics (DLs) and defines important challenges for current system implementations. Recently, several standards for representation languages have been proposed (RDF, OWL). By introducing optimization techniques for inference algorithms we demonstrate that sound and complete query engines for semantic web representation languages can be built for practically significant query classes. The paper introduces and evaluates optimization techniques for the instance retrieval problem w.r.t. the description logic $\mathcal{SHIQ}(\mathcal{D}_n)^-$, which covers large parts of OWL. The paper discusses practical experiments with the description logic system RACER.

Introduction

Practical description logic systems play an ever-growing role for knowledge representation and reasoning research. In particular, the semantic web initiative (Berners-Lee, Hendler, & Lassila 2001) is based on description logics (DLs) and defines important challenges for current system implementations. Recently, one of the main standards for the semantic web has been proposed: the Web Ontology Language (OWL) (van Harmelen *et al.* 2003). OWL is based on two other standards: Resource Description Format (RDF (Lassila & Swick 1999)) and its corresponding "vocabulary language" RDF Schema (RDFS) (Brickley & Guha 2002). In recent research efforts, these languages are mainly considered as ontology representation languages (see e.g. (Baader, Horrocks, & Sattler 2003a) for an overview). The languages are used for defining classes of so-called abstract objects. Now, many applications start to use the RDF part of OWL for representing information about specific abstract objects of a certain domain. Graphical editors such as OilEd (Bechhofer, Horrocks, & Goble 2001) or Protégé (Noy *et al.* 2001) support this way of using OWL quite well.

All information about specific objects (or entities) refers to an ontology (expressed in OWL). Thus, in contrast to, for instance, simple relational databases, queries for retrieving abstract objects described in RDF documents have to be answered w.r.t. to a conceptual domain model (the ontology). The paper introduces and evaluates optimization techniques for the instance retrieval problem w.r.t. the logical basis of OWL DL, the description logic $\mathcal{SHIQ}(\mathcal{D}_n)^-$ (Horrocks, Sattler, & Tobies 2000; Haarslev, Möller, & Wessel 2001), and discusses practical experiments with the description logic system RACER. By introducing optimization techniques for tableau-based inference algorithms we demonstrate that sound and complete query engines for semantic web representation languages can be built for practically significant query classes. The paper is aimed at semantic web systems developers interested in applying and implementing sound and complete knowledge representation and reasoning technologies. Note that we consider soundness and completeness as very important because incompleteness of instance retrieval can even result in unsoundness if the results are used for higher-level purposes in an unreflected way.

The paper presupposes only basic knowledge about description logics (see, e.g., (Baader, Horrocks, & Sattler 2003b)). The interested reader can find a detailed introduction in (Baader *et al.* 2003). A few words about the relationship of description logics, semantic web representation languages, and systems such as RACER are appropriate, however.

RACER reads OWL ontology documents from web servers and represents ontology information as a so-called T-box. T-boxes contain so-called generalized concept inclusions (GCIs). For details about description logic syntax and semantics see, e.g., (Baader *et al.* 2003). RACER accepts the so-called OWL DL subset (van Harmelen *et al.* 2003) with some minimal restrictions such as approximated reasoning for nominals and no full number restrictions for datatype properties (see (Haarslev & Möller 2003) for details). DAML+OIL documents are interpreted with the same restrictions as manifested in OWL DL (van Harmelen *et al.* 2003) (the sets of classes and instances are disjoint, no reified statements, no treatment of class metaobjects etc.). For the results presented in the paper, these restrictions are of no importance.

Descriptions in RDF documents (with OWL DL restrictions) are represented as A-boxes managed by the RACER

System (for details see the RACER User's Guide (Haarslev & Möller 2003)). Basically, the instance retrieval problem for a query concept C_q and an A-box A can be implemented as a sequence of instance tests for all individuals that are mentioned in an A-box. An instance test verifies that an individual i is in the extension of a certain concept in all models of a given T-box and A-box. Retrieving resources described in OWL/RDF documents can be implemented using the A-box instance retrieval inference service (Haarslev & Möller 2001). In this paper we discuss restricted versions of conjunctive queries (Horrocks & Tessaris 2002). An introduction to an XML-based query syntax is given in (Bechhofer, Möller, & Crowther 2003). For presenting examples, however, in this paper, a DL-based syntax is used to ensure readability.

The contribution of the paper is twofold. By introducing and analyzing practical algorithms tested in one of the mature, sound, and complete description logic systems, which is used in many research projects all over the world, the development of even more powerful semantic web query engines is directly supported. All example knowledge-bases we discuss in this paper can be downloaded for verification and comparison purposes (see the RACER download page).

Research Approach, Test Data, and Benchmarks

For implementing sound and complete inference algorithms, tableau-based algorithms are known to provide a powerful basis. Nowadays, almost all practical systems for $\mathcal{SHIQ}(\mathcal{D}_n)^-$ employ highly optimized versions of tableau-based algorithms. It should be emphasized that the research approach behind RACER is oriented towards applications. Thus, we start with optimization techniques for application-specific knowledge bases in order to evaluate optimization techniques in the context of instance retrieval problems. In particular, we consider applications for which A-box reasoning is actually required (i.e., implicit information must be derived from explicit A-box statements, and A-boxes are not only used to store relational data). Thus, instance retrieval cannot be reduced to computing queries for (external) relational databases (see, e.g., (Borgida & Brachman 1993), (Bresciani 1995), (Li & Horrocks 2003)).

An Application-Specific Knowledge Base

For instance, in (Gabsdil, Koller, & Striegnitz 2001a; 2001b) a case-study with the application of DL inference services in a natural language (NL) interpretation system is presented. In particular, the instance retrieval service is investigated for various application-specific subtasks (e.g., resolution of referring expressions, content determination, and content realization). In this application, many A-boxes are generated on the fly (see (Gabsdil, Koller, & Striegnitz 2001a; 2001b) for details) and for each A-box a specific instance retrieval query is computed. Similar approaches are described in (Ludwig, Büchner, & Görz 2002). In order to achieve good performance in the NL application, the performance of the instance retrieval procedure provided by the DL system is crucial. Furthermore, since A-boxes change quite frequently, standard techniques for optimizing instance retrieval using indexing techniques (see below for an explanation) can hardly be employed in order to improve performance because of the overhead of computing index structures in beforehand.

The T-box used in (Gabsdil, Koller, & Striegnitz 2001a; 2001b) consists of 165 possibly cyclic GCIs for concepts as well as domain and range restrictions for 18 roles. In the T-box, many sufficient conditions for concept names are given (with appropriate GCIs, see also declarations with sameConceptAs in OWL). In the A-box around 250 individuals are mentioned in concept and role assertions. The DL used in the knowledge base is a subset of OWL DL (actually, $\mathcal{ALC}$ with inverse roles (Baader *et al.* 2003)).

Synthetic Knowledge Bases for Testing Behavior on Mass Data

For evaluating specific aspects of DL inference engines, a set of benchmarks containing synthetically generated KBs was developed (Motik, Volz, & Maedche 2003). In this paper we consider some of these tests, which are generated automatically due to different strategies. The tests use a T-Box whose concept names form a so-called symmetric concept tree (SCT) w.r.t. the subsumption relation. An SCT is a balanced tree of depth d and branching factor b. For each concept n instances are declared. The instance retrieval query refers to a concept name at the first layer (one of the children of top). The second kind of test is similar to the first one but also declares relations between the individuals (the test is named "SCT rel". An individual is set into relation to a previously generated one via a so-called role assertion (Baader *et al.* 2003). Only one role is used.

The following discussion about optimization techniques starts with insights gained from application-knowledge bases. Later on we use some synthetically generated benchmarks to shed additional light on the behavior of the techniques proposed.

Optimization Techniques and their Evaluation

For applications, which generate A-boxes on the fly as part of their problem-solving processes and ask a few queries w.r.t. each A-box, computing index-structures (with a process called "realization", see below) is not worth the effort. In this section we discuss answering strategies for this kind of application scenario. As we will see, the techniques can also be exploited if index structures are to be computed (possibly off-line).

Transformation of A-Boxes

In order to make realization as fast as possible we investigated ways to maximize the effect of caching techniques supplied by RACER's A-box consistency checking architecture. We transform the original A-box in such a way that acyclic "chains" of roles and individuals are represented by an appropriate exists restriction (see (Haarslev & Möller 2000) for a formal definition of the transformation rules). The corresponding concept and role assertions representing the chains are deleted from the A-box. We illustrate this

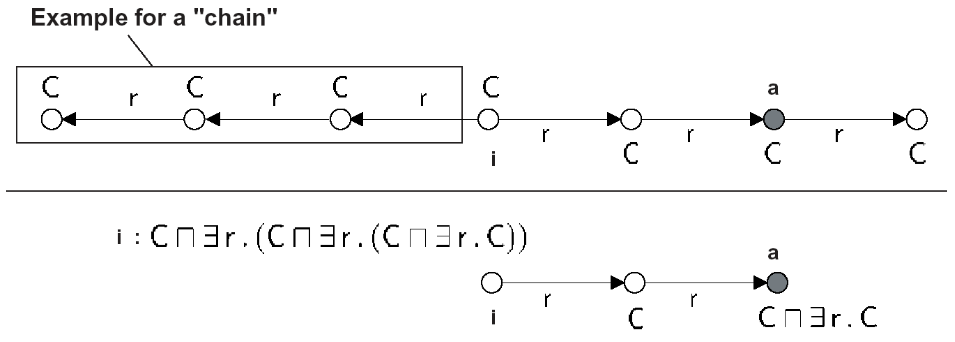

Figure 1: An example for contracting an A-box. The A-box part in the rectangle is replaced with a concept assertion (see the lower part for the resulting A-box).

contraction idea by an example presented in Figure 1. The idea is to transform tree-like role assertions (or "chains") starting' from an individual i into assertions with existential restrictions such that an equisatisfiable A-box is derived (see (Haarslev & Möller 2000) for details). The reason is that in RACER, caching (see also (Haarslev & Möller 2000)) is more effective for concepts rather than for A-box role assertions. Contracting an A-box is part of the processes for building internal data structures for A-box reasoning algorithms.

If an A-box contains individuals that are not "connected" by role assertions (or by constraints involving concrete domains), RACER computes so-called subset A-boxes representing these "islands", applies the algorithms described below to each subset, and combines the results. We do not mention this kind of processing explicitly in the following subsections.

Optimized Linear Instance Retrieval

One possible alternative for implementing instance retrieval is to consider one individual at a time. Hence, the procedure $instance\_retrieval(C_q, A)$ can be implemented by using the following procedure call: $linear\_retrieval(C_q, contract(i, A), individuals(A))$ where $individuals(A)$ returns the set of individuals mentioned in the A-box A and the function $contract$ computes a transformation of an A-box w.r.t. an individual. Except for the contraction idea, linear instance retrieval was also implemented in a similar way in first generation DL systems (see, e.g., (Nebel 1990)).

We assume that $ASAT$ is the standard A-box satisfiability test implemented as an optimized tableau calculus (Horrocks, Sattler, & Tobies 2000; Haarslev, Möller, & Wessel 2001). The function $linear\_retrieval$ is specified by Algorithm 1.

The function call $instance?(i, C, A)$ could be implemented as $\neg ASAT(A \cup \{i : \neg C\})$. However, although this implementation of $instance?$ is sound and complete, it is quite inefficient. A faster variant uses sound but incomplete initial tests for detecting "obvious" non-instances: the individual model merging test (see (Haarslev, Möller, & Turhan 2001))

Algorithm 1 $linear\_retrieval(C, A, candidates)$:

```
$result := \{\}$
for all $ind \in candidates$ do
    if $instance?(ind, C, A)$ then
        $result := result \cup \{ind\}$
    end if
end for
return $result$
```

and a subsumption test involving the negation of the query concept. These two tests (also referred to as "guards" for avoiding to invoke the tableau algorithm) are implemented by Algorithm 2.

Algorithm 2 $obv\_non\_instance?(i, C, A)$:

```
return   $ind\_model\_merge\_poss?(i, neg\_concept(C), A)$
         $\vee\ subsumes?(neg\_concept(C), ind\_concept(i))$
```

Algorithm 3 $subsumes?(C, D)$:

```
if $pmodels\_mergable?(\{cmodel(C), cmodel(D)\})$ then
    return $false$
else
    return $\neg SAT(D \sqcap \neg C)$
end if
```

The subsumption test $subsumes?$ is implemented in Algorithm 3. Algorithm SAT uses a tableau prover for concept consistency (w.r.t. a T-box). Experiences show that, compared to $ASAT$, SAT is usually very fast (although they are of the same worst-case complexity class). The reason is that, at the current state of the art, more optimization techniques have been developed for checking concept consistency (e.g., the trace technique etc.) than for checking A-box consistency. Model merging is a standard optimization technique used in description logic systems (for details see (Horrocks 1997)). It is a fast test for determining whether a concept C does not subsume a concept D. Individual model merging was introduced in (Haarslev, Möller, & Turhan 2001). The main idea of the individual model merging is to extract a (pseudo) model for an individual i from a completion of the A-box A. If the individual model of i and the (pseudo) model of $\neg C$ do not "interact", i can easily be shown not to be an instance of C.

The algorithm $ind\_model\_merge\_poss?$ used in Algorithm 2 is reduced to a pseudo model merging test as follows. It is assumed that $imodel(i, A)$ returns the pseudo model of individual i w.r.t. A and $cmodel(D)$ the pseudo model of concept D.

Algorithm 4 $ind\_model\_merge\_poss?(i, D, A)$:

```
return $pmodels\_mergable?(\{imodel(i, A), cmodel(D)\})$
```

The technique of using an individual model merging test (see (Haarslev, Möller, & Turhan 2001)) is based on the observation that individuals are usually members of only a small number of concepts and the A-boxes resulting from $instance?$ tests are proven as consistent in most cases, i.e., the calls to $instance?$ usually return $false$. The basic idea is to design a cheap but *sound* test for the focused individual i and the concept term $\neg D$ without explicitly considering role assertions and concept assertions for all the other individuals occurring in A. These "interactions" are reflected in the "individual pseudo model" of i. This was the motivation for devising the *individual model merging* technique.

For instance, in the DL $\mathcal{ALC}$ a pseudo model for an individual i mentioned in a consistent initial A-box A w.r.t. a T-box T is defined as follows. Since A is consistent, there exists a set of completions $\mathcal{C}$ of A. Let $A' \in \mathcal{C}$. An *individual pseudo model* M for an individual i in A is defined as the tuple $\langle M^D, M^{\neg D}, M^{\exists}, M^{\forall}\rangle$ w.r.t. A' and A using the following definition.

$$\begin{aligned}
M^D &= \{D \,|\, i : D \in A',\ D \text{ is a concept name}\}\\
M^{\neg D} &= \{D \,|\, i : \neg D \in A',\ D \text{ is a concept name}\}\\
M^{\exists} &= \{R \,|\, i : \exists R.C \in A'\} \cup \{R \,|\, (i, j) : R \in A\}\\
M^{\forall} &= \{R \,|\, i : \forall R.C \in A'\}
\end{aligned}$$

The pseudo model of a concept D is defined analogously by using the completion of an initial A-box $A = \{i : D\}$. Note the distinction between the initial A-box A and its completion A'. Whenever a role assertion exists, which specifies a role successor for the individual i in the initial A-box, the referenced role name is added to the set $M^{\exists}$. This is based on the rationale that the cached pseudo model of i should not refer to individual names occuring already in the initial A-box A. However, it is sufficient to reflect a role assertion $(i, j) : R \in A$ by adding the role name R to $M^{\exists}$. This guarantees that possible interactions via the role R are detected. The algorithm $pmodels\_mergable?$ is defined in Algorithm 5.

Algorithm 5 $pmodels\_mergable?(MS)$

```
return $atoms\_mergable(MS) \wedge roles\_mergable(MS)$
```

The algorithm $atoms\_mergable$ tests for a possible primitive clash between pairs of pseudo models. It is applied to a set of pseudo models MS and returns $false$ if there exists a pair $\{M_1, M_2\} \subseteq MS$ with $(M_1^D \cap M_2^{\neg D}) \neq \emptyset$ or $(M_1^{\neg D} \cap M_2^D) \neq \emptyset$. Otherwise it returns $true$.

The algorithm $roles\_mergable$ tests for a possible role interaction between pairs of pseudo models. It is applied to a set of pseudo models MS and returns $false$ if there exists a pair $\{M_1, M_2\} \subseteq MS$ with $(M_1^{\exists} \cap M_2^{\forall}) \neq \emptyset$ or $(M_1^{\forall} \cap M_2^{\exists}) \neq \emptyset$. Otherwise it returns $true$. The reader is referred to (Haarslev, Möller, & Turhan 2001) for the proof of the soundness of this technique and for further details.

Let us turn back to the function $obv\_non\_instance?$ now. If one of the "guards" in $obv\_non\_instance?$ returns $true$, the result of $instance?$ is $false$. Otherwise, an "expensive" instance test using the tableau algorithm is performed in Algorithm 6. The function $negated\_concept$ used in $obv\_non\_instance?$ returns the negation of its input concept whereas the function $individual\_concept$ returns the conjunction of the concepts in all A-box concept assertions for an individual. For role assertions found in an A-box, we assume additional concept assertions. Role assertions for a role R with i on the left-hand side are represented by at-least terms and, depending on the number of different role assertions for i, corresponding conjuncts $(\geq n\, R)$ are generated by $individual\_concept$. With these auxiliaries, the function $instance?$ can be optimized for the average case but is still sound and complete.

Algorithm 6 $instance?(i, C, A)$:

```
if $obv\_non\_instance?(i, C, A)$ then
  return $false$
else
  return $\neg ASAT(A \cup \{i : \neg C\})$
end if
```

Although this variant of $instance?$ is significantly faster (mainly due to the individual model merging guard), in the Game application discussed above, query answering times in the range of 20 seconds were still unacceptable. Although for many queries the result consists of a set of only very few individuals (compared to 250 individuals mentioned in the A-box) around a hundred individuals still cause the "expensive" $ASAT$ test to be invoked, regardless of the "guards" in Algorithm 6. Thus, although each $ASAT$ test is quite fast (200 milliseconds), its number should be further reduced in order to provide adequate performance.

Binary Instance Retrieval

How can A-box satisfiability tests be avoided at all? The observation is that only very few additions to A of the kind $\{i : \neg C\}$ lead to an inconsistency in the function $instance?$ (i.e., in very few situations i is indeed an instance of C). We found the following procedure to be advantageous.

We assume now that $instance\_retrieval(C_q, A)$ is implemented by calling the procedure $binary\_retrieval(C_q, contract(i, A), individuals(A))$. The function $partition$ is defined in Algorithm 8, it divides a set into two partitions of approximately the same size. Given the partitions, $binary\_retrieval$ calls the function $partition\_retrieval$. The idea of $partition\_retrieval$ (see Algorithm 10) is to

Algorithm 7 $binary\_retrieval(C, A, candidates)$:

```
if $candidates = \emptyset$ then
    return $\emptyset$
else
    $(partition1, partition2) := partition(candidates)$
    return $partition\_retrieval(C, A, partition1, partition2)$
end if
```

first check whether *none* of the individuals in a partition is an instance of the query concept C. This is done with the function $non\_instances?$ (see Algorithm 9).

Algorithm 8 $partition(s)$: /* $s[i]$ refers to the i^{th} element of the set s */

```
if $|s| \leq 1$ then
    return $(s, \emptyset)$
else
    return $(\{s[1], \ldots, s[\lfloor n/2 \rfloor]\}, \{s[\lfloor n/2 \rfloor + 1], \ldots, s[n]\})$
end if
```

Algorithm 9 $non\_instances?(cands, C, A)$:

```
return $ASAT(A \cup \{i : \neg C \mid i \in cands \wedge$
                    $\neg obv\_non\_instance?(i, C, A)\})$
```

The evaluation we conducted with the natural language application indicates that for instance retrieval queries which return only very few individuals a performance gain of up to a factor of 5-10 can be achieved with binary search (compared to linear instance retrieval). The reason is that the $non\_instances?$ test is successful in many cases. Hence, with one "expensive" A-box test a large set of candidates can be eliminated. The underlying assumption is that, in general, the computational costs of checking whether an A-box $(A \cup \{i : \neg C, j : \neg C, \ldots\})$ is consistent is largely dominated by A alone. Hence, it is assumed that the size of the set of constraints added to A has only a limited influence on the runtime. For knowledge bases with, for instance, cyclic GCIs, this may not be the case, however. Partitioning a set of candidates in two parts of approximately the same size can be controlled by heuristics. This has not yet been explored. Thus, further performance gains might be possible.

Dependency-based Instance Retrieval

Although $binary\_retrieval$ is found to be faster in the average case, one can do better. If the function $non\_instances?$ returns $false$, one can analyze the dependencies of the tableau structures ("constraints") involved into all clashes of the tableau branches. Analyzing dependency information for a clash reveals the "original" A-box assertions responsible for the clash. If all clashes are due to an added constraint $i : \neg C$, then, as a by-product of the test, the individual i is known to be an instance of the query concept C. The individual can be eliminated from the set of candidates to be investigated, and it is definitely part of the solution set. Dependency information is kept for other optimization purposes as well (Horrocks 1997) and dependency analysis does not involve much overhead.

Algorithm 10 $partition\_retrieval(C, A, part1, part2)$:

```
if $|part1| = 1$ then
    $\{i\} = part1$
    if $instance?(i, C, A)$ then
        return $\{i\} \cup binary\_retrieval(C, A, part2)$
    else
        return $binary\_retrieval(C, A, part2)$
    end if
else if $non\_instances?(part1, C, A)$ then
    return $binary\_retrieval(C, A, part2)$
else if $non\_instances?(part2, C, A)$ then
    return $binary\_retrieval(C, A, part1)$
else
    return    $binary\_retrieval(C, A, part1)$
              $\cup\ binary\_retrieval(C, A, part2)$
end if
```

Eliminating candidate individuals detected by dependency analysis prevents the reasoner from detecting the same clash over and over again until a partition of cardinality 1 is tested. In the example application, runtimes are reduced by another factor of 3 (compared to binary instance retrieval). If the solution set is large compared to the set of individuals in an A-box, there is some overhead compared to linear instance retrieval because only one individual is removed from the set of candidates at a time as well with the additional cost of collecting dependency information during the tableau proofs. In our investigations, dependency-based instance retrieval was always faster than binary instance retrieval.

Static Index-based Instance Retrieval

The techniques introduced in the previous section can also be exploited if indexing techniques are used for instance retrieval (see, e.g., (Nebel 1990, p. 108f.)). Basically, the idea is to reduce the set of candidates that have to be tested by computing the direct types of every individual. The direct types of an individual i are defined to be the most specific concept names (mentioned in a T-box) of which i is an instance. An index is constructed by deriving a function $associated\_inds$ defined for each concept name C mentioned in the T-box such that $i \in associated\_inds(C)$ iff $C \in direct\_types(i, A)$. Computing the direct types for each individual and the corresponding index $associated\_inds$ is also called *A-box realization*. The optimizations used in the RACER implementation are inspired by the marking and propagation techniques described in (Baader *et al.* 1992; 1994) for exploiting explicitly given information as much as possible.

In the following we assume that CN is the set of all concept names mentioned in the T-box (including the name top). Furthermore, it is assumed that the function $children(C)$ ($parents(C)$) returns the least specific subsumees (most specific subsumers) of C whereas $descendants(C)$ ($ancestors(C)$) returns all subsumees (subsumers) of C. The descendants and ancestors of C include C. Subsumers and subsumees of a concept C are concept names from CN. The function $synonyms(C)$ returns all concept names from CN which are equivalent to C.

The standard way to compute the index is to compute the direct types for each individual mentioned in the A-box separately (one-individual-at-a-time approach). In order to compute the direct types of individuals w.r.t. a T-box and an A-box, the T-box must be classified, i.e., for each concept name mentioned in the T-box (and the A-box) the $parents$ and $children$ are precomputed. Thus, $parents$ and $children$ are not really queries but just functions accessing results stored in data structures. Another view is that the children (or parents) relation defines a lattice whose nodes are concept names (including top and $bottom$). The root node is called top, the bottom node is called $bottom$. This lattice is also referred to as the "taxonomy".

Static index-based instance retrieval was investigated in (Nebel 1990, p. 108f.) and is implemented as follows.

Algorithm 11 $static\_index\_based\_retrieval(C, A)$:

```
if $\exists N \in CN : N \in synonyms(C)$ then
  return $\bigcup_{D \in descendants(C)} associated\_inds(D)$
else
  $known := \bigcup_{D \in descendants(C)} associated\_inds(D)$
  $candidates := \bigcup_{P \in parents(C)}$
                 $\bigcup_{D \in descendants(P)}$
                     $associated\_inds(D)$          (*)
  return $known \cup$
         $instance\_retrieval(C, A, candidates \setminus known)$
end if
```

It is obvious that $instance\_retrieval$ can be implemented by any of the techniques introduced above.

Computing the index structures (i.e., the function $associated\_inds$) is known to be time-consuming. Our findings indicate that for many applications this takes several minutes, i.e. index computation is only possible in a setup phase. Since for many applications this is not tolerable, new techniques had to be developed. The main problem is that for computing the index structure $associated\_inds$ the direct types are computed for every individual in isolation. Rather than asking for the direct types of every individual in a separate query, we investigated the idea of using *sets* of individuals which are "sieved" into the taxonomy, We call the approach the sets-of-individuals-at-a-time approach (see Algorithm 12 and Algorithm 13). The traverse algorithm (Algorithm 12) sets up the index has_member, which is used in $compute\_index\_sets\_of\_inds\_at\_a\_time$ (Algorithm 13). For a given concept name the function has_member returns the instances. The idea of $compute\_index\_sets\_of\_inds\_at\_a\_time$ is to check if the instances of a concept name are not instances of the children of the concept name. Being this the case, the concept name is marked as one of the direct types of each of the instances. This is done by setting the index $associated\_inds$ appropriately.

In the natural language application we investigated, answering a specific query with realization-based instance retrieval and the set-of-individuals-at-a-time approach requires about 30 seconds using dependency-based instance retrieval (and 80 seconds using binary instance retrieval). Thus, for this specific application the performance gain for

Algorithm 12 $traverse(inds, C, A, has\_member)$:

```
if $inds \neq \emptyset$ then
  for all $D \in children(C)$ do
    if $has\_member(D) = unknown$ then
      $instances\_of\_D := instance\_retrieval(D, inds, A)$
      $has\_member(D) := instances\_of\_D$
      $traverse(instances\_of\_D, D, A, has\_member)$
    end if
  end for
end if
```

realization is a factor of three. So it is possible to speed up realization-based instance retrieval to some extent such that queries w.r.t. "static" A-boxes can be processed after quite a short delay. But it still holds that, if A-boxes are not static, i.e., A-boxes are computed on the fly and only very few queries are posed w.r.t. the A-boxes, the direct implementation of instance retrieval as search without exploiting indexes is much faster (and it is possible even without T-box classification).

Algorithm 13 $compute\_index\_sets\_of\_inds\_at\_a\_time(A)$:

```
for all $C \in CN$ do
  $has\_member(C) := unknown$
  $associated\_inds(C) := \emptyset$
end for
$traverse(individuals(A), top, A, has\_member)$;
$has\_member(top) := individuals(A)$
for all $C \in CN$ do
  if $has\_member(C) \neq unknown$ then
    for all $ind \in has\_member(C)$ do
      if $\neg\exists D \in children(C) : ind \in has\_member(D)$
      then
        $associated\_inds(C) :=$
            $associated\_inds(C) \cup \{ind\}$
      end if
    end for
  end if
end for
```

Dynamic Index-based Instance Retrieval

Computing a complete index (realization) as described in the previous subsection is possible if many queries are posed w.r.t. a "fixed" A-box (and T-box). However, sometimes realization is too time-consuming. Therefore, we devised a new strategy that exploits (i) explicitly given information (e.g., from A-box assertions of the form $i : A$ where A is a concept name) and (ii) the results of previous instance retrieval queries.

The idea can be explained as follows. The function $associated\_inds$ associates a set $Inds$ of individuals with each concept name C such that for each $i \in Inds$ it holds that i is an instance of C, for each $D \in descendants(C)$ the individual $i \notin associated\_inds(D)$, and for each $D \in ancestors(C)$ the individual $i \notin associated\_inds(D)$.

The function $associated\_inds$ is updated due to the results of queries. Let us assume $i \in associated\_inds(C)$ and $C \in ancestors(E)$. If it turns out that i is an instance of E, the function $associated\_inds$ is changed accordingly.

Thus, the index evolves as instance retrieval queries are answered. Therefore, we call this strategy dynamic index-based instance retrieval.

In this new approach, the function $associated\_inds(C)$ returns an individual i even if C is not "most specific", i.e. even if there might exist a subconcept D of C such that i is also an instance of D. The consequence is that Algorithm 11 is no longer complete. The idea of only considering the parents of the query concept (see the line marked with an asterisk in Algorithm 11) must be dropped. Before we give a complete algorithm for dynamic index-based instance retrieval, further optimization techniques are introduced.

Let us assume concept D is a subsumer of C. In addition, let us assume in order to answer some previous query the direct types for an individual i are computed. If it is known for an individual $i \in associated\_inds(D)$ that $D \in direct\_types(i)$, then i is removed from the set of candidates for the query concept C. Since D is a subsumer of C and D is a direct type (i.e., D is most specific), i cannot be an instance of C.

With each concept name we also associate a set of non-instances. The non-instances are found by queries for the direct types of an individual (the non-instances are associated with the children of each direct type) or by exploiting previous calls to the function $instance\_retrieval$. If an individual i is found not to be an instance of a query concept D, this is recorded appropriately by including i in $associated\_non\_instance(D)$ if there is no $E \in ancestors(D)$ such that $i \in associated\_non\_instance(E)$ (non-redundant caching). The non-instances of a query concept can then be discarded from the set of candidates. The new algorithm for instance retrieval is shown in Algorithm 14.

Algorithm 14 $dynamic\_index\_based\_retrieval\_1(C, A)$:

```
$known := \bigcup_{D \in descendants(C)} associated\_inds(D)$
$possible\_candidates :=$
    $\bigcup_{D \in (ancestors(C) \setminus \{C\})} associated\_inds(D)$
$candidates := possible\_candidates \setminus$
        $\bigcup_{D \in ancestors(C)} associated\_non\_instances(D)$
return $known \cup$
        $instance\_retrieval(C, A, candidates \setminus known)$
```

Note that instead of testing the parents as done in Algorithm 11 (see the line marked with an asterisk), in Algorithm 14 the descendants of the query concept C are taken into consideration for possible candidates. In other words, it is not a problem if an individual i is returned by $associated\_inds(D)$ although there exist subconcepts of D of which i is also an instance.

In order to evaluate the proposed algorithm, we first use a very simple T-box $\{Article \sqsubseteq Document, Book \sqsubseteq Document, CS\_Book \sqsubseteq Book\}$ and consider an A-box with the following assertions (for n we use different settings):

$doc\_1 : Article, doc\_2 : Article, \ldots, doc\_n : Article,$
$doc\_n{+}1 : Book, doc\_n{+}2 : Book, \ldots, doc\_n{+}n : Book,$
$doc\_n + n + 1 : CS\_Book,$
$doc\_n + n + 2 : CS\_Book,$
$\ldots$
$doc\_n + n + n : CS\_Book$

In order to evaluate Algorithm 14, queries for $Book$ and for CS_Book are executed. Queries can be ordered with respect to subsumption. Given the partial order induced by subsumption, an optimal execution sequence for answering multiple queries can be generated with a topological sorting algorithm. The more general queries are processed first, yielding a (possibly reduced) set of candidates for more specific queries as a by-product. This is demonstrated by considering the query set $\{retrieve((?x), Book(?x)), retrieve((?x), CS\_Book(?x))\}$. There are two strategies, either all instances of CS_Book are retrieved first (Strategy 1) or all instances of $Book$ are retrieved first (Strategy 2). The runtimes of the query sets under different strategies are indicated in Table 1.

n	Gen. Time	ASAT	Strategy 1	Strategy 2
10000	1	6	7	5
20000	3	10	29	19
30000	22	15	79	42
40000	34	23	164	115
50000	54	34	320	200
60000	80	42	904	552

Table 1: Runtimes (in secs) of instance retrieval query sets with different strategies.

In the first column the number n is specified (note that the A-box contains three times as many individuals), in the second column the time to generate the problem (i.e., the time to "fill" the A-box) is specified, in the third column the time for the initial A-box consistency test is displayed, and in the last two columns the runtimes for the different strategies are indicated. All tests were performed on a 1GHz Powerbook running Mac OS X. Memory requirements are neglectable for all experiments ($\leq$ 100MB). Table 1 reveals that for larger values of n, Strategy 2, i.e., to first retrieve all instances of the superconcept $Book$, is approximately twice as fast as Strategy 1. The reason is that with Strategy 2 the set of candidates for the second instance retrieval query can be considerably reduced due to dynamic index-based instance retrieval.

In order to compare static index-based instance retrieval (one-by-one and set-based realization) with dynamic index-based instance retrieval, we used the synthetically generated A-box benchmarks described in the previous section on "Synthetic Knowledge Bases".

The test characteristics are specified in the first four columns (SCT stands for symmetric concept tree, SCT rel stands for symmetric concept tree and individuals associated to one another using relations, see above). In column 'L' the time to load the problem from a file is given, and in column 'B' the time to build the index structures required by consistency checking and instance retrieval is indicated. The column 'ASAT' contains the time for the initial A-box

Name	d	b	n	L	B	ASAT	static (1)	static (2)	dynamic
SCT	3	5	20	0.4	0.5	1.3	6.1	2.7	1.4
SCT	3	5	30	0.5	0.8	2.4	9.9	4.5	2.9
SCT	4	5	10	1.1	1.6	5.4	36.0	7.3	6.2
SCT	4	5	30	3.7	5.1	15.9	330.5	40.7	17.6
SCT	5	5	10	10.9	16.4	18.3	1528.0	70.6	31.7
SCT	5	5	30	62.8	54.8	76.8	timed out	184.7	160.3
SCT rel	3	5	10	0.5	0.8	1.4	2.8	3.6	2.6
SCT rel	4	5	10	3.3	7.5	10.2	40.3	17.4	17.7
SCT rel	5	5	10	22.0	120.0	144.6	1751.0	190.6	159.1

Table 2: Runtimes (in secs) for processing retrieval queries with static and dynamic index-based instance retrieval techniques.

consistency test (including the index building time from column B). The column 'static (1)' indicates the time for instance retrieval using the sets-of-individuals-at-a-time realization approach whereas 'static (2)' indicates the time using the one-individual-at-a-time approach. The last column contains the runtime for dynamic index-based instance retrieval. The results obtained from analyzing the experiments can be summarized as follows.

One-individual-at-a-time realization is much faster for these tests than using sets-of-individuals-at-a-time realization. In these synthetic benchmarks, there exist n instances for each of the b^d concept names. The assumption that the result set contains only few individuals is not met in these benchmarks (the result set contains $b^{(d-1)}$ elements). Furthermore, it can be seen that dynamic index-based instance retrieval causes almost no overhead for these synthetic benchmarks (this may be due to the fact that the retrieval concept is located close to the root of the taxonomy). In addition it becomes apparent that the runtime for instance retrieval is mostly dominated by the initial A-box satisfiability test (which cannot be easily eliminated). In particular, building index structures is an expensive process (see column B) and cannot be neglected. Faster query evaluation results for RACER can be achieved by optimizing this process.

More Expressive Queries

In previous sections we have discusses simple instance retrieval queries. The instances to be retrieved are described by concepts. However, for many application purposes a more expressive query language is appropriate. In this section, an extension to the previously mentioned query language is introduced by discussing several examples. The implementation of the query language in the RACER system is based on the instance retrieval service for which optimization techniques are introduced in this paper. Furthermore, standard optimization techniques for conjunctive queries (Chandra & Merlin 1977) are included.

We assume that the T-box is extended with the following axioms for stating that the range of the function (datatype property) n_copies_sold is $integer$, the inverse of the role (property) has_author is $author\_of$, and some other constraints. For instance, if at least 3000 copies are sold, a $CS\_Document$ becomes a CS_Best_Seller.

$attribute(n\_copies\_sold, integer)$,
$inverse(has\_author, author\_of)$,
$Document \sqsubseteq \exists has\_author.Author$,
$CS\_Document \sqsubseteq Document$,
$min(n\_copies\_sold, 3000) \sqcap CS\_Document \sqsubseteq$
$\quad CS\_Best\_Seller$

The A-box is assumed to be extended with appropriate assertions to specify documents, authors, and sold copies. For instance, $(doc\_1, author\_1) : has\_author$. Now we consider queries over the knowledge base.

- Retrieve the fillers of a role has_author w.r.t. a given individual doc_1:
  $ans(x) \leftarrow has\_author(doc\_1, x)$.
- Retrieve all tuples (x, y) such that x is a $Document$ with author y and at least 5000 copies sold:
  $ans(x, y) \leftarrow$
  $\quad Document(x) \wedge$
  $\quad has\_author(x, y) \wedge$
  $\quad Author(y) \wedge$
  $\quad min(n\_copies\_sold, 5000)(x)$.
- Retrieve all tuples (x, y) such that both x and y are instances of $Author$ and both are $author\_of$ the same book:
  $ans(x, y) \leftarrow$
  $\quad Author(x) \wedge$
  $\quad Author(y) \wedge$
  $\quad Book(z) \wedge$
  $\quad author\_of(x, z) \wedge$
  $\quad author\_of(y, z)$.
- Retrieve all books x for which there does not exist an author:
  $ans(x) \leftarrow$
  $\quad Book(x) \wedge$
  $\quad neg((\exists has\_author.\top)(x))$.
- Retrieve all pairs (x, y) of books and authors for which it cannot be proven that the tuple (x, y) is an element of the relation has_author:
  $ans(x, y) \leftarrow$
  $\quad Book(x) \wedge$
  $\quad Author(y) \wedge$
  $\quad neg(has\_author(x, y))$.

The result of a query is a set of vectors of individuals. Query languages are investigated in detail in (Donini *et al.* 1992;

1996), and we follow this work in assigning a model theoretic semantics to queries. For instance, the semantics of concept predicates such as $Book(?x)$ is defined as the intersection of the extensions of the concept determined for every model. The semantics of two-place predicates such as $has\_author(?x, ?y)$ is defined as the intersection of the extensions of the relation has_author over all models. In other words, in query results variables are only bound to domain elements that are the image of individuals explicitly mentioned in the A-box (in databases we call this the active-domain semantics). If a query is "negated" (operator neg), the complement w.r.t. the domain (unary predicates) or the cross-product of the domain (binary predicates) is indicated. In other words, this corresponds to a negation as failure semantics (see (Donini *et al.* 1992; 1996) for details).

Pragmatic Considerations

It is obvious that in the document retrieval scenario introduced above, only some individuals are expected to be in the result set of an instance retrieval query. In our scenario these "root" individuals are the documents, e.g., doc_1. For other individuals that we could introduce, e.g. authors of books, it might be known in advance that they are only accessed as role fillers of root individuals (see the query examples above). Therefore, in RACER it is possible to explicitly indicate so-called "public" individuals. This is called publishing an individual in RACER terminology. For the example A-box we assume that the following statement is executed:

$publish(\{doc\_1, doc\_2, doc\_3\})$

Published individuals can be returned in the result sets of specific instance retrieval queries (for other queries, e.g. role filler retrieval, publication is not relevant). Using the RACER system, clients can subscribe to a "channel" on which individuals are announced that are instances of a given query concept. As an example we consider a subscription named q_1 with query concept $Book$ and server `"mac1.sts.tu-harburg.de"` running at port 8080. After the query q_1 is registered, RACER running at node `"racer.sts.tu-harburg.de"` sends the following message string to `"mac1.sts.tu-harburg.de"` listening on port 8080: "$((q\_1\ doc\_2)\ (q\_1\ doc\_3))$". Of course, the client is responsible for interpreting the result appropriately. In our agent scenario we assume that two documents, doc_2 and doc_3, are recorded as possible hits to the query (possibly together with retrieved values for the number of copies sold etc.), but we do not go into details here.

Next, we assume that the document information repository represented by the A-box is subsequently filled with information about new documents.

$doc\_4 : CS\_Document$

Much less is known about doc_4, and after publishing it, nobody will be notified. Although there is a subscription to a channel for $Book$, it cannot be proven that doc_4 is an instance of this concept.

Now we assume there are two additional subscriptions q_2 and q_3 to the concepts $CS\_Document$ and CS_Best_Seller, respectively. For query q_2 RACER immediately generates a message $((q\_2\ doc\_4))$ and redirects it to the channel specified with the subscription statement. However, for q_3 no message can be generated at subscription time. As time evolves, it might be the case that the number of copies sold for doc_4 becomes known. The A-box is extended with the following assertions:

$(doc\_4, n\_copies\_4) : n\_copies\_sold,$
$equal(n\_copies\_4, 4000)$

Since the information in the A-box now implies that doc_4 is an instance of the query concept in subscription q_3, the client is notified accordingly using the same techniques as discussed above.

The example sketches how description logics in general, and the publish and subscribe interface of RACER in particular, can be used to implement a document retrieval system (for additional examples and details on the publish and subscribe interface see the RACER User's Guide (Haarslev & Möller 2003)). For answering registered queries, RACER exploits query subsumption as explained above. Furthermore, for incrementally answering queries after additions to an A-box, the set of known results (see, e.g., Algorithm 14) can be extended by exploiting cached results of registered queries.

Conclusion

In this paper we demonstrated optimization techniques that make A-box inferences based on tableau-based DL systems suitable for many non-naive applications. We motivated the techniques described in this paper with the semantic web scenario and its representation language OWL/RDF. In this context, reasoning over individuals (e.g., instance retrieval) cannot be easily reduced to database lookups. The examples we gave here do not cover the full expressivity of OWL, however, they already demonstrate the need for more advanced optimization techniques.

For very restricted sublanguages of OWL (i.e., no existential restrictions at all), initial experiments indicate that datalog-based approaches could become an alternative to tableau-based approaches (Motik, Volz, & Maedche 2003). However, research in this area has just started and no stable implementations are available at the time of this writing. We have shown that tableau-based algorithms provide a sound basis for applications, provided the implementation technique proposed in this paper are implemented in practical systems. Nevertheless, the experiments also show some limitations of current DL technology. Experiences indicate that only up to approximately 30,000 individuals (see Table 1) can be appropriately handled by current system implementations. Further research is necessary (in particular for contexts such as the semantic web) to provide for appropriate internal data structures used in the tableau prover in order to avoid unnecessary overhead for large A-boxes. This will be investigated in future versions of RACER.

Acknowledgments

The work on extended queries was partially supported by the German Science Foundation under grant number NE 279/8-1 and NE-279/8-2. We are grateful to Michael Wessel for incorporating his research on expressive queries into the RACER system. We also thank the anonymous reviewers for valuable comments.

References

Baader, F.; Franconi, E.; Hollunder, B.; Nebel, B.; and Profitlich, H.-J. 1992. An empirical analysis of optimization techniques for terminological representation systems, or: Making KRIS get a move on. In *Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'92)*, 270–281.

Baader, F.; Franconi, E.; Hollunder, B.; Nebel, B.; and Profitlich, H.-J. 1994. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management* 4:109–132.

Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; and Patel-Schneider, P. F., eds. 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Baader, F.; Horrocks, I.; and Sattler, U. 2003a. Description logics as ontology languages for the semantic web. In Hutter, D., and Stephan, W., eds., *Festschrift in honor of Jörg Siekmann*. LNAI. Springer-Verlag.

Baader, F.; Horrocks, I.; and Sattler, U. 2003b. Description logics as ontology languages for the semantic web. In Hutter, D., and Stephan, W., eds., *Festschrift in honor of Jörg Siekmann*, Lecture Notes in Artificial Intelligence. Springer-Verlag. To appear.

Bechhofer, S.; Horrocks, I.; and Goble, C. 2001. OilEd: a reason-able ontology editor for the semantic web. In *Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence, September 19-21, Vienna*. LNAI Vol. 2174.

Bechhofer, S.; Möller, R.; and Crowther, P. 2003. The DIG description interface. In *Proc. International Workshop on Description Logics – DL'03*.

Berners-Lee, T.; Hendler, J.; and Lassila, O. 2001. The semantic web. *Scientific American* May 2001.

Borgida, A., and Brachman, R. 1993. Loading data into description reasoners. *ACM SIGMOD Record* 22(2):217–226.

Bresciani, P. 1995. Querying databases from description logics. In *Knowledge Representation Meets Databases*.

Brickley, D., and Guha, R. 2002. RDF vocabulary description language 1.0: RDF Schema, http://www.w3.org/tr/2002/wd-rdf-schema-20020430/.

Chandra, A. K., and Merlin, P. M. 1977. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Nineth ACM Symposium on Theory of Computing*, 77–90.

Donini, F. M.; Lenzerini, M.; Nardi, D.; Nutt, W.; and Schaerf, A. 1992. Adding epistemic operators to concept languages. In *Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'92)*, 342–353. Morgan Kaufmann, Los Altos.

Donini, F. M.; Lenzerini, M.; Nardi, D.; Nutt, W.; and Schaerf, A. 1996. Adding epistemic operators to description logics. Technical Report 16-96, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza".

Gabsdil, M.; Koller, A.; and Striegnitz, K. 2001a. Building a text adventure on description logic. In *International Workshop on Applications of Description Logics, Vienna, September 18*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-44/`.

Gabsdil, M.; Koller, A.; and Striegnitz, K. 2001b. Playing with description logic. In *Proceedings Second Workshop on Methods for Modalities M4M-02*. `http://turing.wins.uva.nl/~m4m/M4M2/program.html`.

Haarslev, V., and Möller, R. 2000. Consistency testing: The RACE experience. In *Proceedings International Conference Tableaux'2000*, volume 1847 of *Lecture Notes in Artificial Intelligence*, 57–61. Springer-Verlag.

Haarslev, V., and Möller, R. 2001. RACER system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*.

Haarslev, V., and Möller, R. 2003. The Racer user's guide and reference manual.

Haarslev, V.; Möller, R.; and Turhan, A.-Y. 2001. Exploiting pseudo models for tbox and abox reasoning in expressive description logics. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

Haarslev, V.; Möller, R.; and Wessel, M. 2001. The description logic $\mathcal{ALCNH}_{R^+}$ extended with concrete domains: A practically motivated approach. In Goré, R.; Leitsch, A.; and Nipkow, T., eds., *Proceedings of the International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, 2001, Siena, Italy*, Lecture Notes in Computer Science, 29–44. Springer-Verlag.

Horrocks, I., and Tessaris, S. 2002. Querying the semantic web: a formal approach. In Horrocks, I., and Hendler, J., eds., *Proc. of the 13th Int. Semantic Web Conf. (ISWC 2002)*, number 2342 in Lecture Notes in Computer Science, 177–191. Springer-Verlag.

Horrocks, I.; Sattler, U.; and Tobies, S. 2000. Reasoning with individuals for the description logic $\mathcal{SHIQ}$. In MacAllester, D., ed., *Proceedings of the 17th International Conference on Automated Deduction (CADE-17)*, Lecture Notes in Computer Science. Germany: Springer Verlag.

Horrocks, I. 1997. Optimisation techniques for expressive description logics. Technical Report UMCS-97-2-1, University of Manchester, Department of Computer Science.

Lassila, O., and Swick, R. 1999. Resource description framework (RDF) model and syntax specification. recommendation, W3C, february 1999. http://www.w3.org/tr/1999/rec-rdf-syntax-19990222.

Li, L., and Horrocks, I. 2003. Matchmaking using an instance store: Some preliminary results. In *Proceedings of the 2003 International Workshop on Description Logics (DL'2003)*.

Ludwig, B.; Büchner, K.; and Görz, G. 2002. Mapping semantics onto pragmatics. In Görz, G.; Haarslev, V.; Lutz, C.; and Möller, R., eds., *KI-2002 Workshop on Applications of Description Logics, CEUR Proceedings, Aachen*.

Motik, B.; Volz, R.; and Maedche, A. 2003. Optimizing query answering in description logics using disjunctive deductive databases. In *Proceedings of the 10th International Workshop on Knowledge Representation Meets Databases (KRDB-2003)*, 39–50.

Nebel, B. 1990. *Reasoning and Revision in Hybrid Representation Systems*, volume 422 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

Noy, N. F.; Sintek, M.; Decker, S.; Crubezy, M.; Fergerson, R. W.; and Musen, M. A. 2001. Creating semantic web contents with Protege-2000. *IEEE Intelligent Systems* 16(2):60–71.

van Harmelen, F.; Hendler, J.; Horrocks, I.; McGuinness, D. L.; Patel-Schneider, P. F.; and Stein, L. A. 2003. OWL web ontology language reference, http://www.w3.org/tr/owl-guide/.