J Autom Reasoning manuscript No.
(will be inserted by the editor)

On the Scalability of Description Logic Instance Retrieval

Volker Haarslev · Ralf Möller

Received: date / Accepted: date

Abstract Practical description logic systems play an ever-growing role for knowledge representation and reasoning research even in distributed environments. In particular, the ontology layer of the often-discussed semantic web is based on description logics (DLs) and defines important challenges for current system implementations. The article 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 the Web Ontology Language (OWL). We demonstrate that sound and complete query engines for OWL-DL can be built for practically significant query classes. Experience with ontologies derived from database content has shown that it is often necessary to effectively solve instance retrieval problems with respect to huge amounts of data descriptions that make up major parts of ontologies. We present and analyze the main results about how to address this kind of scalability problem.

Keywords Description logics · Instance retrieval

1 Introduction

Practical description logic systems play an ever-growing role for knowledge representation and reasoning research. In particular, the ontology layer of the semantic web [7] 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) [50]. Although implemented description logics become more and more expressive (e.g., [40]), studies [56,51] have revealed that scalability w.r.t. DL expressiveness is still an ongoing research topic. Besides expressiveness, scalability it is also necessary to be able to deal with huge amounts of data descriptions very effectively. Thus, in practice, description logic systems offering high expressivity must also be able to handle

Volker Haarslev
Department of Computer Science & Software Engineering, Concordia University, Montreal, Quebec, Canada
E-mail: haarslev@cse.concordia.ca

Ralf Möller
Hamburg University of Technology, Hamburg, Germany
E-mail: r.f.moeller@tu-harburg.de

large bulks of assertional knowledge (also referred to as data descriptions) possibly derived from database content. Users expect that DL systems scale w.r.t. data descriptions.

We consider two kinds of scalability: scalability w.r.t. large sets of data descriptions (data description scalability), i.e., runtimes scale well with increased data sizes but unchanged conceptual descriptions, and scalability w.r.t. increased expressivity (expressivity scalability), i.e., a reasoner can still process an ontology in reasonable time if the data size remains unchanged but more complex conceptual descriptions (e.g., full negation, disjunction) are added.

In the literature, the data description scalability problem has been tackled from different perspectives. We see two main approaches, the layered approach and the integrated approach. In the layered approach the goal is to use databases for storing and accessing data, and exploit description logic ontologies for convenient query formulation. The main idea here is to support ontology-based query translation to relational query languages (SQL, Datalog). See, for example, DLDB [28], Instance Store [6], deductive databases [52], or DL-Lite [11]. We notice that these approaches are only applicable if reduced expressivity for conceptual descriptions is acceptable. Despite the most appealing argument of reusing database technology (in particular services for persistent data), at the current state of the art it is not clear how expressivity can be increased to, e.g., $\mathcal{SHIQ}$, without losing the applicability of transformation approaches (e.g., [10]). Hence, while data description scalability is achieved, it is not clear how to extend these approaches to achieve expressivity scalability (at least for some parts of the data descriptions).

We employ the integrated approach because it allows us to investigate solutions to both the expressivity and data description scalability problem. This approach, on the other hand, addresses query answering with a tableau-based description logic system augmented with new techniques inspired from database systems. We note that this approach needs a solution to persistency, i.e., how can a tableau-based system support persistent knowledge bases and inference results without sacrificing its performance too much, but we do not further address this issue here.

This article introduces and evaluates optimization techniques for instance retrieval w.r.t. the logical basis of OWL-DL (without nominals in concept descriptions), and discusses practical experiments with the description logic system RACERPRO[1] (version 1.9.1) which is based on the RACER architecture [31]. The description logic $\mathcal{SHIQ}(\mathcal{D}_n)^-$ implemented by RACERPRO is based on the union of $\mathcal{SHN}(\mathcal{D}_n)^-$ [33] and $\mathcal{SHIQ}$ [37]. By introducing optimization techniques for tableau-based algorithms we demonstrate that sound and complete query engines for practically significant query classes of OWL-DL can be built.

In this article we also discuss grounded conjunctive queries where all variables are so-called distinguished (or must-bind) variables [19], i.e., variables occurring in a query range over the individuals mentioned in the Abox. Due to feedback from hundreds of users of RACERPRO, we know that the grounded conjunctive queries we investigate in this article already cover many important application scenarios for description logics. Specific classes of conjunctive queries with non-distinguished variables can be reduced to queries with distinguished variables by using a so-called rolling-up technique that is applied as a preprocessing step.Work on general conjunctive queries is described in [12,38,41,23].

This article presupposes basic knowledge of description logics and related tableau methods (see, e.g., [3]). The next section gives a brief overview of the description logic $\mathcal{SHIQ}$

[1] RACERPRO is freely available for research and educational purposes (`http://www.racer-systems.com`).

Syntax	**Semantics**
Concepts	
A	$\mathsf{A}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, A is a concept name
$\neg\mathsf{C}$	$\Delta^{\mathcal{I}} \setminus \mathsf{C}^{\mathcal{I}}$
$\mathsf{C} \sqcap \mathsf{D}$	$\mathsf{C}^{\mathcal{I}} \cap \mathsf{D}^{\mathcal{I}}$
$\mathsf{C} \sqcup \mathsf{D}$	$\mathsf{C}^{\mathcal{I}} \cup \mathsf{D}^{\mathcal{I}}$
$\exists\,\mathsf{R}\,.\,\mathsf{C}$	$\{a \in \Delta^{\mathcal{I}} \mid \exists\, b \in \Delta^{\mathcal{I}} : (a,b) \in \mathsf{R}^{\mathcal{I}},\ b \in \mathsf{C}^{\mathcal{I}}\}$
$\forall\,\mathsf{R}\,.\,\mathsf{C}$	$\{a \in \Delta^{\mathcal{I}} \mid \forall\, b : (a,b) \in \mathsf{R}^{\mathcal{I}} \Rightarrow b \in \mathsf{C}^{\mathcal{I}}\}$
$\exists_{\geq n}\,\mathsf{S}\,.\,\mathsf{C}$	$\{a \in \Delta^{\mathcal{I}} \mid \lVert\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in \mathsf{S}^{\mathcal{I}},\ b \in \mathsf{C}^{\mathcal{I}}\}\rVert \geq n\}$
$\exists_{\leq m}\,\mathsf{S}\,.\,\mathsf{C}$	$\{a \in \Delta^{\mathcal{I}} \mid \lVert\{b \in \Delta^{\mathcal{I}} \mid (a,b) \in \mathsf{S}^{\mathcal{I}},\ b \in \mathsf{C}^{\mathcal{I}}\}\rVert \leq m\}$
Roles	
R	$\mathsf{R}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
R^{-}	$\{(a,b) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (b,a) \in \mathsf{R}^{\mathcal{I}}\}$

Terminological Axioms	
Syntax	Satisfied if
$\mathsf{R} \in T$	$\mathsf{R}^{\mathcal{I}} = (\mathsf{R}^{\mathcal{I}})^{+}$
$\mathsf{R} \sqsubseteq \mathsf{T}$	$\mathsf{R}^{\mathcal{I}} \subseteq \mathsf{T}^{\mathcal{I}}$
$\mathsf{C} \sqsubseteq \mathsf{D}$	$\mathsf{C}^{\mathcal{I}} \subseteq \mathsf{D}^{\mathcal{I}}$

Assertional Axioms	
Syntax	Satisfied if
$\mathsf{a} : \mathsf{C}$	$\mathsf{a}^{\mathcal{I}} \in \mathsf{C}^{\mathcal{I}}$
$(\mathsf{a},\mathsf{b}) : \mathsf{R}$	$(\mathsf{a}^{\mathcal{I}}, \mathsf{b}^{\mathcal{I}}) \in \mathsf{R}^{\mathcal{I}}$
$\mathsf{a} \doteq \mathsf{b}$	$\mathsf{a}^{\mathcal{I}} = \mathsf{b}^{\mathcal{I}}$
$\mathsf{a} \not\doteq \mathsf{b}$	$\mathsf{a}^{\mathcal{I}} \neq \mathsf{b}^{\mathcal{I}}$

Fig. 1 Syntax and Semantics of $\mathcal{SHIQ}$ ($n, m \in \mathbb{N},\ n > 1, m > 0$, $\lVert \cdot \rVert$ denotes set cardinality, S is a simple role).

and DL-based inference services. The interested reader can find a more detailed discussion in the DL Handbook [4].

The contribution of this article is twofold. By introducing and analyzing practical *instance retrieval algorithms* tested in one of the most mature, sound, and complete description logic systems (see also [51] for an evaluation of different DL reasoners and their reliability and scalability), the development of better scalable semantic web query engines is directly supported. This part is partially based on [32] but combines and extends the previously reported results with new optimization techniques and new insights derived from further application ontologies. On the other hand the contribution also presents and analyzes the main results we have found about how to start solving the scalability problem with tableau-based prover systems given large sets of data descriptions for a large number of individuals and *grounded conjunctive queries*. The optimization techniques might very well be appropriate for general conjunctive queries. The part on grounded conjunctive queries is partially based on [43] but provides some new techniques and a far more comprehensive evaluation.

2 The Description Logic $\mathcal{SHIQ}$

For the sake of completeness and readability we briefly introduce the description logic $\mathcal{SHIQ}$ [37] using a standard Tarski-style semantics based on an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is the non-empty domain and $\cdot^{\mathcal{I}}$ the interpretation function. The interpretation $\mathcal{I}$ gives meaning to the atomic constructs of $\mathcal{SHIQ}$ and is extended to complex constructs as shown in Figure 1. We assume a set of concept names CN and a set of role names RN. The disjoint subsets P and T of RN denote non-transitive and transitive roles, respectively ($RN = P \cup T$).

If $\mathsf{R}, \mathsf{T} \in RN$ are role names, then the terminological axiom $\mathsf{R} \sqsubseteq \mathsf{T}$ is called a *role inclusion axiom*. A *role hierarchy* $\mathcal{R}$ is a finite set of role inclusion axioms. In order to preserve decidability a syntactic restriction holds for the combinability of number restrictions and transitive roles in $\mathcal{SHIQ}$. Number restrictions are only allowed for simple roles, a role is called *simple* if it is neither transitive nor has any transitive sub-role (see [37] for details). The concept name $\top$ ($\bot$) is used as an abbreviation for $\mathsf{A} \sqcup \neg\mathsf{A}$ ($\mathsf{A} \sqcap \neg\mathsf{A}$) for any $\mathsf{A} \in CN$.

If C and D are concept expressions, then $\mathsf{C} \sqsubseteq \mathsf{D}$ (*generalized concept inclusion* or *GCI*) is a terminological axiom. A finite set of terminological axioms $\mathcal{T}$ is called a *terminology* or *Tbox* w.r.t. to a given role hierarchy $\mathcal{R}$.[2] We use $\mathsf{C} \equiv \mathsf{D}$ as an abbreviation for $\{\mathsf{C} \sqsubseteq \mathsf{D}, \mathsf{D} \sqsubseteq \mathsf{C}\}$.

Let C be a concept expression, R be a role name, O be the set of individual names (disjoint from the set of concepts names and the set of role names), $\mathsf{a}, \mathsf{b} \in O$ be individual names, then $\mathsf{a} : \mathsf{C}$ is called an *instance assertion*, $(\mathsf{a}, \mathsf{b}) : \mathsf{R}$ a *role assertion*, and $\mathsf{a} \doteq \mathsf{b}$ ($\mathsf{a} \not\doteq \mathsf{b}$) an *individual equality (disjointness) assertion.* A finite set of assertional axioms $\mathcal{A}$ w.r.t. a Tbox $\mathcal{T}$ and a role hierarchy $\mathcal{R}$ is called an *Abox*.

Based on these definitions we introduce a set of inference services for concept expressions. All these inference services are defined relative to a given Tbox $\mathcal{T}$ and a role hierarchy $\mathcal{R}$, i.e., whenever we mention some interpretation $\mathcal{I}$ in this section, we assume that $\mathcal{I}$ satisfies $\mathcal{T}$ and $\mathcal{R}$. In the following, let A and B be concept names and C and D be arbitrary complex concepts.

- Concept satisfiability: Given a concept C, does an interpretation $\mathcal{I}$ exist such that $\mathsf{C}^{\mathcal{I}} \neq \emptyset$.
- Concept subsumption: Given two concepts C and D, do all interpretations $\mathcal{I}$ satisfy $\mathsf{C}^{\mathcal{I}} \subseteq \mathsf{D}^{\mathcal{I}}$ (C is subsumed by D or D subsumes C).
- Concept equivalence: Given two concepts C and D, does it hold that C subsumes D and D subsumes C.
- Determine the parents and children of a given concept C: The parents (children) of C are the most specific (general) concept names in CN which subsume (are subsumed by) C. More formally:
  $parents(\mathsf{C}) := \{\mathsf{A} \in CN \cup \{\top\} \mid$ for all models $\mathcal{I}$ of the Tbox it holds that $\mathsf{C}^{\mathcal{I}} \subset \mathsf{A}^{\mathcal{I}}$ and there does not exist a $\mathsf{B} \in CN$: $\mathsf{C}^{\mathcal{I}} \subset \mathsf{B}^{\mathcal{I}} \wedge \mathsf{B}^{\mathcal{I}} \subset \mathsf{A}^{\mathcal{I}}\}$.
  $parents(\mathsf{C}) := \{\mathsf{A} \in CN \cup \{\bot\} \mid$ for all models $\mathcal{I}$ of the Tbox it holds that $\mathsf{A}^{\mathcal{I}} \subset \mathsf{C}^{\mathcal{I}}$ and there does not exist a $\mathsf{B} \in CN$: $\mathsf{A}^{\mathcal{I}} \subset \mathsf{B}^{\mathcal{I}} \wedge \mathsf{B}^{\mathcal{I}} \subset \mathsf{C}^{\mathcal{I}}\}$.
- Tbox classification: Compute the so-called “taxonomy” of a Tbox $\mathcal{T}$, a lattice defined by the children relation between concept names where $\top$ is the root and $\bot$ the bottom node.

The following inference services are defined relative to a given Abox $\mathcal{A}$ which “depends” on a given $\mathcal{T}$ and $\mathcal{R}$, i.e., whenever we mention some interpretation $\mathcal{I}$, we assume that $\mathcal{I}$ satisfies $\mathcal{A}$, $\mathcal{T}$, and $\mathcal{R}$.

- Check the consistency of an Abox: Given an Abox $\mathcal{A}$, does an interpretation $\mathcal{I}$ exist which satisfies all assertions in $\mathcal{A}$. Or in other words: Are the restrictions given in $\mathcal{A}$ too strong, i.e., do they cause a contradiction. Other queries are only possible w.r.t. a given consistent Abox.
- Instance testing: Given an individual a and a concept C, does an interpretation $\mathcal{I}$ exist which satisfies $\mathsf{a}^{\mathcal{I}} \in \mathsf{C}^{\mathcal{I}}$. In other words, is a an instance (or member) of C. This test can be reduced to checking whether the Abox $\mathcal{A} \cup \{\mathsf{a} : \neg\mathsf{C}\}$ is inconsistent.
- Instance retrieval: Find all individuals in O that are instances of a given concept C: $instances(\mathsf{C}) := \{\mathsf{a} \in O \mid$ for all models I it holds that $\mathsf{a}^{\mathcal{I}} \in \mathsf{C}^{\mathcal{I}}\}$.

[2] The reference to $\mathcal{R}$ is omitted in the following.

- Direct types: For a given individual a find the most specific concept names of which a is an instance: $\mathit{direct\_types}(\mathsf{a}) := \{\mathsf{A} \in \mathit{CN} \mid$ for all models I it holds that $\mathsf{a} \in \mathsf{A}^{\mathcal{I}}$ and there does not exist a $\mathsf{B} \in \mathit{CN}$: $\mathsf{a}^{\mathcal{I}} \in \mathsf{B}^{\mathcal{I}}$ and $\mathsf{B}^{\mathcal{I}} \subset \mathsf{A}^{\mathcal{I}}\}$.
- Abox realization: Compute for all individuals mentioned in $\mathcal{A}$ their corresponding set of direct types.
- Role fillers: $\mathit{role\_fillers}(\mathsf{a}, \mathsf{R}) := \{\mathsf{b} \in O \mid$ for all models I it holds that $(\mathsf{a}^{\mathcal{I}}, \mathsf{b}^{\mathcal{I}}) \in \mathsf{R}^{\mathcal{I}}\}$. Note that this might not just be a trivial lookup in the Abox because $\mathcal{SHIQ}$ supports role hierarchies, transitive roles, and number restrictions.

For brevity, for an Abox assertion α we also write $(\mathcal{T}, \mathcal{A}) \models \alpha$ if α is satisfied by all models of $\mathcal{T}$ and $\mathcal{A}$.

3 General Optimizations

There are some general, well known, optimization techniques that we employ. To make this article more self-sufficient, we describe them briefly in this section.

3.1 Individual Pseudo Model Merging

In this part we review the individual pseudo model merging technique [35]. The technique of using an individual model merging test is based on the observation that individuals are usually members of only a small number of concepts, and the Aboxes used as input for instance tests are proven as consistent in most cases. This was the motivation for devising the *individual pseudo model merging* technique. The basic idea is to have a fast *sound* but possibly incomplete test to check whether a focused individual i is an instance of a concept term D without the need to explicitly consider role and concept assertions for all other individuals occurring in $\mathcal{A}$. These possible "interactions" are reflected in the definition of an "individual pseudo model" of i (see below).

For instance, in the DL $\mathcal{ALC}$ a pseudo model for an individual i mentioned in a consistent initial Abox $\mathcal{A}$ w.r.t. a Tbox $\mathcal{T}$ is defined as follows. Since $\mathcal{A}$ is consistent, there exists a set of completions $\mathcal{C}$ of $\mathcal{A}$ (again, see [35] for an exact definition). Let $\mathcal{A}' \in \mathcal{C}$. An *individual pseudo model* M for an individual i in $\mathcal{A}$ is defined as the tuple $\langle M^{\mathsf{A}}, M^{\neg\mathsf{A}}, M^{\exists}, M^{\forall}\rangle$ w.r.t. $\mathcal{A}'$ as follows.

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

The pseudo model of a concept D is defined analogously by using a completion of an initial Abox $\mathcal{A} = \{\mathsf{i}:\mathsf{D}\}$.

Whenever a role assertion exists which specifies a role successor for an individual i in the initial Abox, the referenced role name is added to the set $M^{\exists}$. Cached individual pseudo models should not refer to other individuals in order to be compatible with concept pseudo models. Thus, it is sufficient to reflect a role assertion $(\mathsf{i},\mathsf{j}):\mathsf{R} \in \mathcal{A}$ by adding the role name

Algorithm 1 $pmodels\_mergable?(M_1, M_2)$

```
return $atoms\_mergable(M_1, M_2) \wedge roles\_mergable(M_1, M_2)$
```

R to $M^{\exists}$. This guarantees that possible interactions via the role R are reflected. The function $pmodels\_mergable?$ is defined in Algorithm 1.

The algorithm $atoms\_mergable$ tests for a possible concept interaction between a pair of pseudo models. It is applied to these pseudo models and returns *false* if $(M_1^{\mathsf{A}} \cap M_2^{\neg\mathsf{A}}) \neq \emptyset$ or $(M_1^{\neg\mathsf{A}} \cap M_2^{\mathsf{A}}) \neq \emptyset$. Otherwise it returns $true$.

The algorithm $roles\_mergable$ tests for a possible role interaction between a pair of pseudo models. It returns *false* if $(M_1^{\exists} \cap M_2^{\forall}) \neq \emptyset$ or $(M_1^{\forall} \cap M_2^{\exists}) \neq \emptyset$. Otherwise it returns $true$. The reader is referred to [35] for the proof of the soundness of this technique and for further details.

The algorithm $ind\_model\_merge\_poss?$ (used in the following sections) can be reduced to a pseudo model merging (see Algorithm 2 below). It is assumed that $imodel(\mathsf{i}, \mathcal{A})$ returns the pseudo model of individual i w.r.t. $\mathcal{A}$ and $cmodel(\mathsf{D})$ the pseudo model of concept D.

Algorithm 2 $ind\_model\_merge\_poss?(\mathsf{i}, \mathsf{D}, \mathcal{A})$:

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

Individual pseudo model merging can be easily extended to $\mathcal{SHIQ}$ by additionally considering role hierarchies, number restrictions, transitive and functional roles. For the sake of brevity we do not present these extensions in this article.

3.2 Individual Concept

Based an a given Abox (or completion) $\mathcal{A}$ one can define a so-called individual concept. It is defined as a concept derived from all instance and role assertions in $\mathcal{A}$ where this individual occurs, i.e., $individual\_concept(\mathsf{i}, \mathcal{A}) = \sqcap(\{\mathsf{C} \mid \mathsf{i} : \mathsf{C} \in \mathcal{A}\} \cup \{\exists_{\geq 1}\, \mathsf{R} \mid (\mathsf{i}, \mathsf{j}) : \mathsf{R} \in \mathcal{A}\} \cup \{\exists_{\geq 1}\, \mathsf{R}^{-} \mid (\mathsf{j}, \mathsf{i}) : \mathsf{R} \in \mathcal{A}\})$. If the unique name assumptions holds, $\exists_{\geq n}\, \mathsf{R}$ and $\exists_{\geq n}\, \mathsf{R}^{-}$ are included where n depends on the number of different individual names j_k occurring in $(\mathsf{i}, \mathsf{j}_k) : \mathsf{R}$ and $(\mathsf{j}_k, \mathsf{i}) : \mathsf{R}$. By definition, an individual i is an instance of its associated individual concept. Individual concepts are used later (see Section 5.3 and 5.4) as individual representatives in retrieval inference services.

3.3 GCI Absorption and Lazy Unfolding

Another standard optimization technique, GCI absorption [36,39,57], tries to transform GCIs in an satisfiability-preserving way in order to facilitate the application of the lazy unfolding technique [2]. In general, a Tbox can be divided into two sets of axioms. The set $\mathcal{T}_U$ contains axioms of the form $\mathsf{A} \sqsubseteq \mathsf{C}$ or $\neg\mathsf{A} \sqsubseteq \mathsf{D}$ respectively, where A is a concept name and C and D are concept expressions. The concept names occurring on the left-hand side of the axioms in $\mathcal{T}_U$ are called *unfoldable*. The second set $\mathcal{T}_G$ contains axioms of the form $\mathsf{C} \sqsubseteq \mathsf{D}$ where C and D are concept expressions (see [39,57] for a more detailed analysis of GCI absorption).

The *lazy unfolding* technique works as follows. Whenever a tableau algorithm encounters an assertion of the form $\mathsf{a} : \mathsf{A}$ ($\mathsf{a} : \neg\mathsf{A}$) for the first time in an Abox, and an axiom of the form $\mathsf{A} \sqsubseteq \mathsf{C}$ ($\neg\mathsf{A} \sqsubseteq \mathsf{D}$) can be found in $\mathcal{T}_U$, it adds $\mathsf{a} : \mathsf{C}$ ($\mathsf{a} : \mathsf{D}$) to the Abox. A terminology is called *cyclic* if the unfolding of a concept name A during a concept satisfiability test might result in unfolding A again.

The GCI absorption technique tries to maximize the effectiveness of lazy unfolding by transforming axioms in $\mathcal{T}_G$ in such a way that they can be moved to $\mathcal{T}_U$. If possible, the set $\mathcal{T}_G$ should become empty after the application of GCI absorption. If axioms remain in $\mathcal{T}_G$, they have to be considered as possible disjunctions for every individual encountered during a concept satisfiability or Abox consistency test (again, see [39,57] for more details). The number of remaining axioms in $\mathcal{T}_G$ is usually a good indication for the hardness of a given Tbox. Some of the optimization techniques presented in the following sections rely on subsumption tests which might become expensive if $\mathcal{T}_G$ is not empty after the absorption preprocessing step.

4 Research Approach and Benchmark Philosophy

We evaluated the optimization techniques presented in the following sections in the context of Abox realization and instance retrieval problems for application knowledge bases. In particular, we consider applications for which Abox reasoning is actually required, i.e., implicit information must be derived from Abox statements and Tbox axioms, and Aboxes are not only used to store relational data. Thus, instance retrieval cannot be reduced to computing queries for (external) relational databases (see, e.g., [8], [10], [6]).

We do not compare query answering speed with other DL systems but instead investigate the effect of optimization techniques that could be exploited by any (tableau-based) DL inference system that already exists or might be built. From a methodological point of view, performance comparisons with other systems (e.g., see [44]) are not as informative as one might think. The reason is that, in general, it is hard to operate systems in the same mode with optimization techniques switched on and off. In addition, whether a certain system seems to be slow for some specific knowledge base and query, might be the result of various effects that can hardly be tracked down from an external point of view, and those effects tell us nothing about the usefulness of the optimization techniques under investigation.

Recently, various optimization techniques for partitioning Aboxes into independent parts and/or creating condensed (summary) Aboxes [20,26,16] have been reported. The advantages of Abox partitioning are not the topic of this investigation. RACER employs a straightforward Abox partitioning technique [30] which is based on pure connectedness of graphs because an Abox can be viewed as a possibly cyclic graph defined by a set of role assertions. More precisely, if an Abox can be partitioned into independent parts which are also not related via assertions involving concrete domains, RACER employs a divide-and-conquer strategy which applies the algorithms described below to each partition and combines the results afterwards.

We would like to emphasize that the optimization techniques reported in this article are still very useful in the presence of more refined Abox partition schemes because they are applicable to single partitions. Moreover, our techniques are even more vital for scenarios where Aboxes cannot be partitioned or contain large partitions.

For the evaluation we selected a set of 10 application knowledge bases (see also Section 7.1 for more details) with usually small and simple Tboxes but large Aboxes whose sizes are varying between 700 and 18K individuals, 9K and 51K individual assertions, and between

650 and 88K role assertions. Furthermore we tested three ontologies with very large Aboxes. LUBM [29] is used with two different Tboxes (LUBM-lite and LUBM) and 6 different Abox sizes (5 to 50 universities) which result for 50 universities in 1 082K individuals, 3 355K individual assertions, and 3 298K role assertions. UOBM [42] was derived from LUBM. It exhibits a more complicated Tbox. The Aboxes we investigated contain up to 138K individuals, 509K individual assertions, and 563K role assertions. Wordnet (version 1.7.1) is an OWL-DL KB representing the WordNet 1.7.1 lexical database and contains 84K concept names, 269K individual assertions, 548K individual assertions, and 304K role assertions.

All these ontologies are tested with various sets of optimization settings which disable or enable particular optimization techniques that are introduced in the following sections. In general, we use two scenarios for the evaluation: (i) Abox realization and (ii) instance retrieval or, more general, query answering (without precomputing a realization).

5 Optimization Techniques for Instance Retrieval

In this section we first review known optimization techniques and then present novel techniques that are partially based on or extend these known techniques. We discuss answering strategies for instance retrieval that do not rely on Abox realization, which usually takes a long time. However, as we will see later, these techniques can also be exploited if index structures are to be computed. For applications, which either generate Aboxes on the fly as part of their problem-solving processes and/or ask a few queries w.r.t. an Abox, computing index structures by realization might not always be worth the effort (see also Section 7 for evidence supporting this heuristic). We like to point out that some of the techniques proposed in this section take advantage of a precomputed taxonomy, i.e., some of the proposed indexing techniques become more effective if the Tbox has already been classified. The only technique which requires a classified Tbox as prerequisite is static index-based instance retrieval (introduced in Section 5.7). In case the Tbox has not been classified, it is assumed that all concept names have $\top$ as parent and $\bot$ as child.

5.1 Transformation of Aboxes

In order to make Abox reasoning as fast as possible we investigated ways to maximize the effect of caching techniques. The efficacy of caching techniques from a theoretical and practical point of view is well known (e.g., see [34, 17, 14, 15, 25]). We transform the original Abox in such a way that acyclic "chains" formed by role assertions are represented by appropriate existential restrictions (see [34] for a formal definition of the transformation rules). The corresponding concept and role assertions representing the chains are deleted from the Abox. Given a particular inference service request for selected individuals, the idea is to transform tree-like role assertions (or "chains") starting from these individuals into assertions with existential restrictions such that an equisatisfiable Abox is derived to answer this request (see [34] for details). These contractions need to be computed on demand and depend on the involved individuals and the requested Abox inference services.

We illustrate this contraction idea by an example presented in Figure 2. If one assumes the query whether the individual i is an instance of $\exists\,\mathsf{r}_1\,.\,\mathsf{C}_2$, the contracted Abox in the lower part of Figure 2 is sufficient to answer this query. Rolling-up techniques developed for conjunctive query answering (e.g., [22]) apply a similar transformation. The contraction

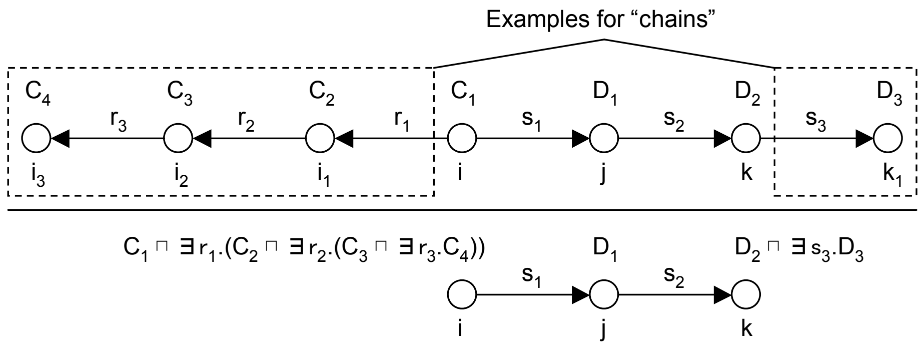


Fig. 2 An example for contracting an Abox. The Abox parts in the rectangles are replaced with concept assertions (see the lower part for the resulting Abox).

can be useful to maximize reuse of caches about the satisfiability status of existential concept expressions. Contracting an Abox is part of the process to build internal data structures for Abox reasoning and it can be potentially expensive. Currently, this transformation is only applied to Aboxes which, considering the Tbox, use the language $\mathcal{ALC}$. The procedure, as specified here, is not applicable to DLs with number restrictions.

5.2 Linear Instance Retrieval

The following sections (5.2-5.8) have the goal to present various optimization techniques to implement the Abox inference service of retrieving *all* instances for a given query concept C and an Abox $\mathcal{A}$: $instance\_retrieval(\mathsf{C}, \mathcal{A})$. However, from a methodological point of view it is advantageous to provide a lower-level inference service that only tests a specified set of individuals (called *candidates*) whether they are instances of C: $instance\_retrieval(\mathsf{C}, \mathcal{A},$ $candidates)$. Whenever we use $instance\_retrieval(\mathsf{C}, \mathcal{A})$ in the following sections (without specifying a value for $candidates$), it is assumed that this is used as an abbreviation for $instance\_retrieval(\mathsf{C}, \mathcal{A}, individuals(\mathcal{A}))$ where $individuals(\mathcal{A})$ returns the set of individuals mentioned in the Abox $\mathcal{A}$.

A traditional way to implement instance retrieval is to iterate over all known individuals. Hence, $instance\_retrieval(\mathsf{C}, \mathcal{A}, candidates)$ can be rewritten as $linear\_retrieval(\mathsf{C}, \mathcal{A},$ $individuals(\mathcal{A}))$. The function $linear\_retrieval$ collects all instances of a concept C from a set of candidates as specified by Algorithm 3. Except for the contraction idea, linear instance retrieval has already been implemented in a similar way in first generation DL systems (see, e.g., [46]).

Algorithm 3 $linear\_retrieval(\mathsf{C}, \mathcal{A}, candidates)$:

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

The function $contract(\mathsf{i}, \mathcal{A})$ computes the contracted variant of Abox $\mathcal{A}$ w.r.t. the individual i. In the following we assume that SAT ($ASAT$) is the standard concept satisfiability (Abox consistency) test implemented by an optimized tableau calculus [37,33].

5.3 Obvious Non-Instances: Exploiting Completion Information

The function call $instance?(\mathsf{i}, \mathsf{C}, \mathcal{A})$ could be reduced to $\neg ASAT(\mathsf{A} \cup \{\mathsf{i} : \neg \mathsf{C}\})$. However, although this implementation of $instance?$ is sound and complete, it is quite inefficient. A faster variant is given in Algorithm 4. It uses sound but incomplete initial tests for detecting "obvious" non-instances: the individual pseudo model merging test (see [35] and Section 3.1) and the Abox completion test (see below). If one of the "guards" in $instance?$ returns $true$, the result of $instance?$ is $false$. Otherwise the result of an appropriate Abox inconsistency test is returned. In this way the function $instance?$ is optimized for the average case but remains sound and complete.

Algorithm 4 $instance?(\mathsf{i}, \mathsf{C}, \mathcal{A})$:

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

The Algorithms 5 and 7 define two tests which are used in Algorithm 4 as guards with the goal to avoid the use of $ASAT$.

Algorithm 5 $obv\_non\_instance?(\mathsf{i}, \mathsf{C}, \mathcal{A})$:

```
if $ind\_model\_merge\_poss?(\mathsf{i}, negated\_concept(\mathsf{C}), \mathcal{A})$ then
    return $true$
else if use_subsumption_test_on_negated_concept then
    return $subsumes?(negated\_concept(\mathsf{C}), individual\_concept(\mathsf{i}, \mathcal{A}))$
else
    return $false$
```

The function $obv\_non\_instance?$ calls the individual pseudo model merging test with the negated query concept and possibly a subsumption test checking whether the negated query concept subsumes the individual concept. The function $negated\_concept$ returns the negation of its input concept. The $individual\_concept$ is defined in Section 3.2. The subsumption test $subsumes?$ is defined in Algorithm 6. For some KBs (with simple Tboxes) a SAT test with an individual concept is often faster compared to a corresponding $ASAT$ test (although both are of the same worst-case complexity). The reason is that better optimization techniques are available for SAT than for $ASAT$ tests. However, if the GCIs contained in an KB are "difficult", i.e., enforce a high amount of runtime, the concept subsumption test in Algorithm 6 might become too expensive. A good indicator seems to be the set of axioms which could not be absorbed (see Section 3.3). If this set is not empty, proofs tend to be complex. Our findings suggest to initialize the global parameter '$\mathrm{use\_subsumption\_test\_on\_negated\_concept}$' with $false$ by default.

Algorithm 6 $\mathit{subsumes?}(\mathsf{C}, \mathsf{D})$:

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

Let us turn back to the function $\mathit{instance?}$ now. Although the call of $\mathit{obv\_non\_instance?}$ is often successful, we devised another non-instance test reusing information from a saved Abox completion. The function $\mathit{completion\_ASAT}$ (see Algorithm 7) invokes ASAT but uses a saved completion $\mathcal{A}'$ instead of the original input Abox $\mathcal{A}$. If $\mathit{completion\_ASAT}$ is unsuccessful, an "expensive" instance test using the original Abox $\mathcal{A}$ is performed as shown in Algorithm 4.

Algorithm 7 $\mathit{completion\_ASAT}(\mathsf{i}, \mathsf{C}, \mathcal{A})$

```
return $ASAT(completion(\mathcal{A}) \cup \{\mathsf{i} : \neg \mathsf{C}\})$
```

The function $\mathit{completion}(\mathcal{A})$ returns an associated completion $\mathcal{A}'$ for an Abox $\mathcal{A}$. The completion technique is sound but incomplete. If the completion $\mathcal{A}'$ extended by the assertion $\mathsf{i} : \neg \mathsf{C}$ is satisfiable, then the individual i is obviously not an instance of C. However, if the union of the completion $\mathcal{A}'$ and $\{\mathsf{i} : \neg \mathsf{C}\}$ is unsatisfiable, an ASAT test, where the original input Abox $\mathcal{A}$ is extended, might still find a different completion $\mathcal{A}''$ that additionally satisfies the assertion $\mathsf{i} : \neg \mathsf{C}$.

Although for many instance retrieval queries the result set often consists of a small set of individuals, some individuals might still cause the "expensive" ASAT test to be invoked, regardless of the "guards" in Algorithm 4. Thus, the number of ASAT tests should be further reduced in order to improve the performance of instance retrieval.

5.4 Obvious Instances: Exploiting Precompletion Information

This observation leads to another optimization technique with the goal to find "obvious" instances with minimum effort. This can improve data description scalability, for example for ontologies with large Aboxes where many individuals are obvious instances of concepts. For reasons of brevity, an element of a completion is called a *constraint* in the following.

Given an initial Abox consistency test and a completion, we assume that each element of a completion is marked with a set of dependency constraints which were used to derive the element. Let us illustrate this with a very simple example. Given an Abox (with an empty Tbox) $\mathcal{A} = \{(\mathsf{a}, \mathsf{b}) : \mathsf{R},\ \mathsf{a} : \forall \mathsf{R}.\mathsf{C}\}$ the tableau rule for universal restrictions would create the completion $\{\langle (\mathsf{a}, \mathsf{b}) : \mathsf{R}, \emptyset \rangle, \langle \mathsf{a} : \forall \mathsf{R}.\mathsf{C}, \emptyset \rangle, \langle \mathsf{b} : \mathsf{C}, \{\langle (\mathsf{a}, \mathsf{b}) : \mathsf{R}, \emptyset \rangle, \langle \mathsf{a} : \forall \mathsf{R}.\mathsf{C}, \emptyset \rangle\}\rangle\}$. The dependency set for $\mathsf{b} : \mathsf{C}$ consists of the constraints (and their dependencies) required to trigger the tableau rule for universal restrictions.

Given a completion $\mathcal{C}$, a *precompletion* is now defined as the set $\mathcal{C} \setminus \{\langle c, E \rangle \in \mathcal{C} \mid E \neq \emptyset$ and $\mathit{con}(\mathit{dep}(E))$ contains a concept of the form $\mathsf{C} \sqcup \mathsf{D}$ or $\exists_{\leq n} \mathsf{R} . \top$ or $\exists_{\leq m} \mathsf{R} . \mathsf{C}\}$ with $n > 1$ and $m \geq 1$. We use two auxiliary functions $\mathit{dep}(E) = E \cup \bigcup_{\langle c, E' \rangle \in E} \mathit{dep}(E')$ and $\mathit{con}(E) = \{\mathsf{C} \mid \langle c, E' \rangle \in E$ and E' contains an assertion of the form $\langle \mathsf{a} : \mathsf{C}, E'' \rangle\}$. The constraints of a precompletion are also referred to as *deterministic* constraints in the following.

Given the precompletion constraints, for an individual i an approximation of its most-specific concept (MSC) can be computed (the approximation is called MSC'). The generation of MSC' is identical to the generation of an individual concept (see Section 3.2) but it is based on a precompletion instead of a completion. If MSC' is subsumed by a query concept C, then i must be an instance of C.

Our analysis of KBs from the semantic web community showed that quite a few of these KBs have large Aboxes (e.g., LUBM or InfoGlue) which contain assertions that lead to many deterministic constraints in tableau proofs and, thus, large precompletions. This results in the fact that for many instances of a query concept C (e.g., for LUBM the concept $\mathsf{Faculty}$ used in query $Q9$, see also Section 6) the instance problem is decided with a subsumption test based on the MSC' of each individual. Subsumption tests are known to be usually fast due to caching and pseudo model merging. The more precisely MSC' approximates MSC, the more often an individual can be determined to be an obvious instance of a query concept. Clearly, it might be possible to determine obvious instances by directly considering the precompletion data structures. However, at this implementation level a presentation would be too detailed. The main point is that, due to our findings, the crude approximation with MSC' suffices to solve many instance tests in KBs such as LUBM.

If concept names are used for query optimization (e.g., in the case of LUBM), a large number of tests for obvious non-instances or obvious instances can determine the result (see also Section 6.1). However, for some individuals i and query concepts C both tests might not determine whether i is (not) an instance of C (e.g., this is the case for the concept Chair in LUBM). Since both of these "cheap" tests are incomplete, for these individuals (e.g., i) a refutational Abox consistency test, where the assertion $\mathsf{i} : \neg\mathsf{C}$ has been added to the Abox, must be decided with a sound and complete tableau prover. For some concepts, the set of candidates might become quite large. Considering the volume of assertions in LUBM (see Section 7.3.2 for details), it is easy to see that the Abox consistency test should not start from the initial, unprocessed Abox in order to ensure scalability.

For large Aboxes and many repetitive instance tests it is a waste of resources to "expand" the very same initial constraints over and over again. Therefore, the precompletion resulting from the initial Abox consistency test is used as a starting point for refutational instance tests. The tableau prover keeps the precompletion in memory. If some constraint is added, only a limited amount of work needs to be done.

5.5 Binary Instance Retrieval

Despite the techniques explained in the previous sections, in some cases refutational Abox consistency tests on single individuals are required. How can these Abox consistency tests be avoided or at least their number be reduced? The observation for many KBs is that only very few additions to $\mathcal{A}$ of the kind $\{\mathsf{i} : \neg\mathsf{C}\}$ lead to an inconsistency in the function $instance?$ (i.e., in very few situations i is indeed an instance of C). Thus, it might be advantageous to combine several individual (non-)instance tests into one Abox consistency test based on a standard divide-and-conquer strategy. This scheme is based on splitting a set of individuals into binary partitions for instance retrieval as shown in Algorithm 8.

We assume in the following that $instance\_retrieval(\mathsf{C}, \mathcal{A})$ is implemented by calling $binary\_retrieval(\mathsf{C}, \mathcal{A}, individuals(\mathcal{A}))$. The function $partition$ is defined in Algorithm 9. It divides a set of individuals into two partitions of approximately the same size. Given the partitions, $binary\_retrieval$ calls $partition\_retrieval$. The idea of $partition\_retrieval$ (see

Algorithm 8 $binary\_retrieval(\mathsf{C}, \mathcal{A}, candidates)$:

```
if $candidates = \emptyset$ then
    return $\emptyset$
else
    $\langle partition_1, partition_2\rangle := partition(candidates)$
    return $partition\_retrieval(\mathsf{C}, \mathcal{A}, partition_1, partition_2)$
```

Algorithm 11) is to 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 10).

Algorithm 9 $partition(s)$: /* $s[i]$ refers to the i^{th} element of the set s */

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

Algorithm 10 $non\_instances?(cands, \mathsf{C}, \mathcal{A})$:

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

The potential performance gain is based on the observation that the $non\_instances?$ test is successful in many cases. Hence, with one "expensive" Abox test a large set of candidates can be eliminated. The underlying assumption is that, in general, the computational costs of checking whether an Abox $(\mathcal{A} \cup \{\mathsf{i} : \neg\mathsf{C},\ \mathsf{j} : \neg\mathsf{C}, \ldots\})$ is consistent is largely dominated by $\mathcal{A}$ alone. Hence, it is assumed that the size of the set of constraints added to $\mathcal{A}$ has only a limited influence on the runtime. For knowledge bases with, for instance, terminological cycles or a non-empty set $\mathcal{T}_G$ of unabsorbed 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 fully explored. Thus, further performance gains might be possible.

Algorithm 11 $partition\_retrieval(\mathsf{C}, \mathcal{A}, part_1, part_2)$:

```
if $|part_1| = 1$ then
    let i be the only member of $part_1$
    if $instance?(\mathsf{i}, \mathsf{C}, \mathcal{A})$ then
        return $\{\mathsf{i}\} \cup binary\_retrieval(\mathsf{C}, \mathcal{A}, part_2)$
    else
        return $binary\_retrieval(\mathsf{C}, \mathcal{A}, part_2)$
else if $non\_instances?(part_1, \mathsf{C}, \mathcal{A})$ then
    return $binary\_retrieval(\mathsf{C}, \mathcal{A}, part_2)$
else if $non\_instances?(part_2, \mathsf{C}, \mathcal{A})$ then
    return $binary\_retrieval(\mathsf{C}, \mathcal{A}, part_1)$
else
    return $binary\_retrieval(\mathsf{C}, \mathcal{A}, part_1) \cup binary\_retrieval(\mathsf{C}, \mathcal{A}, part_2)$
```

5.6 Dependency-based Instance Retrieval

Although $binary\_retrieval$ is found to be faster than linear retrieval 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 in all clashes of the tableau branches. Analyzing dependency information for a clash reveals the “original” Abox assertions responsible for the clash. If the clashes in all attempts to construct a completion are due to an added constraint $\mathsf{i} : \neg\mathsf{C}$, 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 of the query. Dependency information is kept for other optimization purposes as well [36] and dependency analysis does not involve much overhead.

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. If the solution set is large compared to the set of individuals in an Abox, 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.

5.7 Static Index-based Instance Retrieval

The techniques introduced in the previous sections can also be exploited if indexing techniques are used for instance retrieval (see, e.g., [46, p. 108f]). Basically, the idea is to reduce the set of candidates that have to be tested by computing the direct types (see Section 2) of every individual. An index is constructed by specifying a function $associated\_inds$ applicable to each concept name A mentioned in the Tbox such that $\mathsf{i} \in associated\_inds(\mathsf{A})$ iff $\mathsf{A} \in direct\_types(\mathsf{i}, \mathcal{A})$. The optimizations techniques for maximally exploiting explicitly given information are inspired by the marking and propagation techniques described in [2].

5.7.1 One Individual at a Time

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

The standard way to compute the index is to compute the direct types for each individual mentioned in the Abox separately (one-individual-at-a-time approach). In order to compute the direct types of individuals w.r.t. a Tbox and an Abox, the Tbox must be classified first (see Section 2). Static index-based instance retrieval was investigated in [46, p. 108f.] and a variation of it is implemented as shown Algorithm 12. The input parameter $supplied\_candidates$ can be used to restrict the set of candidates. In case $supplied\_candidates$ is empty, it is set to the set all of individuals known in the given Abox. The default value for the parameter $supplied\_candidates$ is the empty set.

One might be tempted to simplify the line marked by an asterisk and only consider the parents of the query concept A. Thus, this line would be changed to the following:

Algorithm 12 $static\_index\_based\_retrieval(\mathsf{A}, \mathcal{A}, supplied\_candidates)$:

```
if $\exists\, \mathsf{N} \in CN : \mathsf{N} \in synonyms(\mathsf{A})$ then
    return $\bigcup_{\mathsf{B} \in descendants(\mathsf{A})} associated\_inds(\mathsf{B})$
else
    $known := \bigcup_{\mathsf{B} \in descendants(\mathsf{A})} associated\_inds(\mathsf{B})$
    if $supplied\_candidates = \emptyset$ then
        $supplied\_candidates := individuals(\mathcal{A})$
    $candidates := \bigcup_{\mathsf{P} \in parents(\mathsf{A})} \bigcup_{\mathsf{B} \in descendants(\mathsf{P})} associated\_inds(\mathsf{B}) \cap$        (*)
                   $supplied\_candidates$
    return $known \cup instance\_retrieval(\mathsf{A}, \mathcal{A}, candidates \setminus known)$
```

$candidates := \bigcup_{\mathsf{P} \in parents(\mathsf{A})} associated\_inds(\mathsf{B}) \cap supplied\_candidates$. However, the following counterexample exists where some candidates would be ignored by this simplification. Let us assume the Tbox $\mathcal{T} = \{\mathsf{A} \equiv \exists\, \mathsf{R} \,.\, \mathsf{X}, \mathsf{D} \equiv \exists\, \mathsf{R} \,.\, \mathsf{Y}, \mathsf{Y} \sqsubseteq \mathsf{X}\}$ and the given query concept $\mathsf{Q} := \exists\, \mathsf{R} \,.\, \mathsf{X} \sqcap \exists_{\geq 1}\, \mathsf{S}$. Given this Tbox the set of parents for Q is $\{\mathsf{A}\}$. Obviously, Q is not subsumed by D. Now let us consider the Abox $\mathcal{A} = \{(\mathsf{i}, \mathsf{j}) : \mathsf{R}, (\mathsf{i}, \mathsf{j}) : \mathsf{S}, \mathsf{j} : \mathsf{Y}\}$. It is easy to see that it holds that $\mathsf{i} \in associated\_inds(\mathsf{D})$. Hence, the simplification to use only the associated individuals of the parents to compute the candidates would not consider instances of D and thus the "simplification" would make Algorithm 12 incomplete.

It is obvious that $instance\_retrieval$ used in Algorithm 12 can be implemented by any of the techniques introduced in the previous sections.

5.7.2 Sets of Individuals at a Time

Computing the index structures (i.e., the function $associated\_inds$) is known to be time-consuming. Our findings (see also Section 7) indicate that for many applications this might take several minutes or even hours, i.e., index computation is often 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 looping over all individuals and asking for the direct types of each individual in a separate query, we investigated the idea of using *sets* of individuals which are "sieved" into the taxonomy. We call this approach the sets-of-individuals-at-a-time approach (see Algorithms 14 and 13).

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

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

The traverse procedure (Algorithm 14) sets up the index has_member. For a concept name A the function has_member is used to specify the set of individuals which are "known" to be instances of A. The procedure $compute\_index\_sets\_of\_inds\_at\_a\_time$ (Algorithm 13)

Algorithm 14 $traverse(inds, \mathsf{A}, \mathcal{A}, has\_member)$:

```
if $inds \neq \emptyset$ then
  for all $\mathsf{B} \in children(\mathsf{A})$ do
    if $has\_member(\mathsf{B}) = \text{unknown}$ then
      $instances\_of\_B := instance\_retrieval(\mathsf{B}, inds, \mathcal{A})$
      $has\_member(\mathsf{B}) := instances\_of\_B$
      $traverse(instances\_of\_B, \mathsf{B}, \mathcal{A}, has\_member)$
```

uses has_member in order to check if the instances of a concept name are *not* instances of the children of the concept name. If this is the case, the concept name is marked as one of the direct types of each of the instances. This is done by setting up the index $associated\_inds$.

5.8 Dynamic Index-based Instance Retrieval

Precomputing a complete index (realization) as described in the previous subsection is reasonable if many queries (using concept names) are posed w.r.t. a "fixed" Abox (and Tbox). However, realization is often too time-consuming if only a few queries are executed afterwards. Therefore, we investigated a new strategy that exploits (i) explicitly given information (e.g., from Abox assertions of the form $\mathsf{i}:\mathsf{A}$ where A is a concept name) and (ii) the results of previous instance retrieval queries. This dynamic index resembles realization because it also focuses on concept names but it is created and maintained lazily. In contrast to static index-based retrieval, the version in this section does not depend on a classified Tbox but its performance can be improved by an already existing taxonomy.

The idea is to have a function $dyn\_associated\_inds$ which associates a set of individuals with each concept name A such that for each $\mathsf{i} \in Inds$ it holds that (i) i is an instance of A, (ii) $\mathsf{B} \in (descendants(\mathsf{A}) \cup ancestors(\mathsf{A})) \Rightarrow \mathsf{i} \notin dyn\_associated\_inds(\mathsf{B})$.

The function $dyn\_associated\_inds$ is updated due to the results of queries. Let us assume $\mathsf{i} \in dyn\_associated\_inds(\mathsf{A})$ and $\mathsf{A} \in ancestors(\mathsf{E})$. If it turns out that i is an instance of E, the function $dyn\_associated\_inds$ is changed accordingly. Thus, the index evolves as instance retrieval queries are answered. Therefore, we call this strategy dynamic index-based instance retrieval.

In contrast to $associated\_inds$ from the previous section, $dyn\_associated\_inds(\mathsf{A})$ returns an individual i even if A is not "most specific", i.e., even if there might exist a yet unknown subconcept B of A such that i is also an instance of B. The consequence is that Algorithm 12 is no longer complete. The idea of only considering the parents (and their descendants) of the query concept (see the line marked with an asterisk in Algorithm 12) must be dropped to regain completeness. Before we give a complete algorithm for dynamic index-based instance retrieval, further optimization techniques are introduced.

Let us assume the concept B is a subsumer of A. In addition, let us assume the direct types of an individual i had been previously computed for answering some query. If it is known for an individual $\mathsf{i} \in dyn\_associated\_inds(\mathsf{B})$ that $\mathsf{B} \in direct\_types(\mathsf{i})$, then i is removed from the set of candidates for the query concept A. Since B is a subsumer of A and B is a direct type (i.e., B is most specific), i cannot be an instance of A.

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 B, this is recorded appropriately by including i in $dyn\_associated\_non\_instance(\mathsf{B})$ provided there does not exists a concept $\mathsf{E} \in \mathit{ancestors}(\mathsf{B})$ such that $\mathsf{i} \in \mathit{dyn\_associated\_non\_instance}(\mathsf{E})$ (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 15. It has a parameter $\mathit{supplied\_candidates}$ which can be used to extend the set of possible candidates. In case $\mathit{supplied\_candidates}$ is empty, it is set to the set of all individuals known in the given Abox. The default value for the parameter $\mathit{supplied\_candidates}$ is the empty set.

Algorithm 15 $\mathit{dynamic\_index\_based\_retrieval}(\mathsf{A}, \mathcal{A}, \mathit{supplied\_candidates})$:

```
$\mathit{known} := \bigcup_{\mathsf{B} \in \mathit{descendants}(\mathsf{A})} \mathit{dyn\_associated\_inds}(\mathsf{B})$
**if** $\mathit{supplied\_candidates} = \emptyset$ **then**
    $\mathit{supplied\_candidates} := \mathit{individuals}(\mathcal{A})$
$\mathit{possible\_candidates} := \bigcup_{\mathsf{B} \in (\mathit{ancestors}(\mathsf{A}) \setminus \{\mathsf{A}\})} \mathit{dyn\_associated\_inds}(\mathsf{B}) \cup$
                            $\mathit{supplied\_candidates}$
$\mathit{candidates} := \mathit{possible\_candidates} \setminus \bigcup_{\mathsf{B} \in \mathit{ancestors}(\mathsf{A})} \mathit{dyn\_associated\_non\_instances}(\mathsf{B})$
**return** $\mathit{known} \cup \mathit{instance\_retrieval}(\mathsf{A}, \mathcal{A}, \mathit{candidates} \setminus \mathit{known})$
```

Note that instead of testing the descendants of the parents as done in Algorithm 12 (see the line marked with an asterisk), in Algorithm 15 the ancestors of the query concept A are taken into consideration for possible candidates. In other words, it is not a problem if an individual i is returned by $\mathit{dyn\_associated\_inds}(\mathsf{B})$ although there exist subconcepts of B of which i is also an instance.

5.9 OWL-DL Datatype Properties

RACER supports concrete domain reasoning as defined in [33] and reasoning about fillers of OWL-DL datatype properties can be easily mapped on concrete domain reasoning using appropriate concrete domains for strings, booleans, integers, and reals. However, this mapping turned out to be too expensive for datatype properties unless some of them are restricted by number restrictions. RACER analyzes submitted KBs and dynamically enables a special and simpler reasoning method for datatype properties if possible. Fillers of datatype properties adhere to a locality restriction, e.g., if two different individuals i and j have fillers "*joe*" and "*bill*" for an OWL-DL datatype property has_name, the reasoning remains local w.r.t. i and j respectively. In other words: there cannot exist an interaction between the datatype property filler of i and j. This optimization technique seems to be quite effective for very large Aboxes such as LUBM containing many assertions about datatype properties.

5.10 Indexing of Role Assertions

Tableau algorithms do not apply a rule to assertions of the form $\mathsf{i} : \exists \mathsf{R}.\mathsf{C}$ if there already exists a R-role filler for i that is known to be an instance of concept C. Index structures are required to efficiently determine whether there already exists a corresponding role filler. With index structures maintained efficiently, a role filler lookup can be implemented in almost constant time.

6 Optimizations for Grounded Conjunctive Queries

In addition to the basic concept-based instance retrieval inference service, more expressive query languages are required in practical applications. Well-established is the class of conjunctive queries. A *conjunctive query* consists of a *head* and a *body*. A query is a structure of the form $ans(X_1, \ldots, X_n) \leftarrow atom_1, \ldots, atom_m$. The head lists variables for which the user would like to compute bindings. The body consists of query atoms (see below) in which all variables from the head must be mentioned. If the body contains additional variables, they are seen as existentially quantified. A query answer is a set of tuples representing bindings for variables mentioned in the head.

Query atoms can be *concept* query atoms ($C(X)$), *role* query atoms ($R(X, Y)$), *same-as* query atoms ($X = Y$) as well as so-called *concrete domain* query atoms. The latter are introduced to provide support for querying the concrete domain part of a knowledge base and will not be covered in detail here. Conjunctive queries are built from query atoms using boolean constructs for conjunction (indicated with comma) or union ($\vee$).

In *standard* conjunctive queries, variables (in the head and in query atoms in the body) are bound to (possibly anonymous) domain objects. A system supporting (unions of) standard conjunctive queries is QuOnto [1]. In so-called *grounded* conjunctive queries, $C(X)$, $R(X, Y)$ or $X = Y$ are true if, given some bindings α for mapping from variables to *individuals mentioned in the Abox* $\mathcal{A}$, it holds that $(\mathcal{T}, \mathcal{A}) \models \alpha(X) : C$, $(\mathcal{T}, \mathcal{A}) \models (\alpha(X), \alpha(Y)) : R$, or $(\mathcal{T}, \mathcal{A}) \models \alpha(X) \doteq \alpha(Y)$, respectively. In grounded conjunctive queries the standard semantics can be obtained for so-called tree-shaped queries by using corresponding existential restrictions in query atoms. Due to space restrictions, we cannot discuss the details here. In the following, we consider only grounded conjunctive queries. For more information on RACER's Abox query language we refer to [53,54]. The language is called nRQL (pronounce: "niracle" and hear it as "miracle").

We use the LUBM benchmark for illustrative purposes in this section. LUBM queries are modeled as grounded conjunctive queries referencing concept, role, and individual names from the Tbox. Below, the LUBM queries 9 and 12 are shown in order to demonstrate LUBM query answering problems – note that 'www.University0.edu' is an individual and *subOrganizationOf* is a transitive role. Please refer to [27–29] for more information about the LUBM queries.

$$\begin{aligned}
Q9 : ans(x, y, z) \leftarrow\ & Student(x), Faculty(y), Course(z), \\
& advisor(x, y), takesCourse(x, z), teacherOf(y, z) \\
Q12 : ans(x, y) \leftarrow\ & Chair(x), Department(y), memberOf(x, y), \\
& subOrganizationOf(y, \text{www.University0.edu})
\end{aligned}$$

In order to investigate the data description scalability problem, we used the Tbox provided with the LUBM benchmarks. The Tbox declares (and sometimes uses) inverse and transitive roles as well as domain and range restrictions, but no number restrictions, value restrictions or disjunctions. Among other axioms, the LUBM Tbox contains axioms that express necessary and sufficient conditions for some concept names. For instance, the Tbox contains an axiom for Chair: $\mathsf{Chair} \equiv \mathsf{Person} \sqcap \exists \mathsf{headOf}.\mathsf{Department}$. For evaluating our optimization techniques for query answering we consider runtimes for a whole query set (queries 1 to 14 in the LUBM case).

If grounded conjunctive queries are answered in a naive way by evaluating subqueries in the sequence of syntactic notation, acceptable answering times can hardly be achieved.

For efficiently answering queries, a query execution plan is determined by a cost-based optimization component (c.f., [21, p. 787ff.], see also [13]) which orders query atoms such that queries can be answered effectively. Query execution plans are specified here in the same notation as queries (whether a query is seen as an execution plan will be clear from context). We assume that in query execution plans the execution order of atoms is determined by the order in which they are textually specified.

Let us consider the execution plan $ans(x, y) \leftarrow C(x), R(x, y), D(y)$. Processing the atoms from left to right will start with the atom $C(x)$. Since there are no bindings known for the variable x, the atom $C(x)$ is mapped to a query $instance\_retrieval(\mathsf{C}, \mathcal{A}, individuals(\mathcal{A}))$. The elements in the result set of the retrieval query are possible bindings for x. $C(x)$ is called a *generator*. The next query atom in the execution plan is $R(x, y)$. There are bindings known for x but no bindings for y. Thus, $R(x, y)$ is also a generator (for y-bindings). Given the atom $R(x, y)$ is handled by a role filler query for each binding of x, there are possible bindings generated for y. Afterwards, the atom $D(y)$ is treated. Since there are bindings for y available, the atom is mapped to an instance test (for each binding). We say, the atom $D(y)$ acts as a *tester*.

Determining all bindings for a variable (with a generator) is much more costly than verifying a particular binding (with a tester). Treating the one-place predicates $Student$, $Faculty$, and $Course$ from query $Q9$ (see above) as generators for bindings for corresponding variables results in a combinatorial explosion (cross product computation). Optimization techniques are required to provide for efficient query answering in the average case.

6.1 Query Optimization

The optimization techniques that we investigated are inspired by database join optimizations, and, for example, exploit the fact that there are few $Faculty$ members but many $Students$ in the LUBM data descriptions. For instance, in case of query $Q9$, the idea is to use $Faculty$ as a generator for bindings for y and then generate the bindings for z following the role $teacherOf$. The heuristics applied here is that the average cardinality of a set of role fillers is rather small. For the given z bindings we apply the predicate $Course$ as a tester (rather than as a generator as in the naive approach). Given the remaining bindings for z, bindings for x can be established via the inverse of $takesCourse$. These x bindings are then filtered with the tester $Student$.

If z was not mentioned in the set of variables for which bindings are to be computed (in the head of the query), and the tester $Course$ was not used, there would be no need to generate bindings for z at all. One could just check for the existence of a $takesCourse$ role filler for bindings w.r.t. x. This way, further optimizations are possible.

In the second example, query $Q12$, the constant (individual) '$\mathrm{www.University0.edu}$' is mentioned. Starting from this individual the inverse of the role $subOrganizationOf$ is applied as a generator for bindings for y which are filtered with the tester $Department$. With the inverse of $memberOf$, bindings for x are computed which are then filtered with $Chair$. Since for the concept $Chair$ sufficient conditions are declared in the Tbox, instance retrieval reasoning is required if $Chair$ is a generator. Thus, it is advantageous that $Chair$ is applied as a tester (and only instance tests are performed).

For computing a query execution plan, a total order relation on query atoms with respect to a given set of data descriptions (assertions in an Abox) is required. For determining the order relation, we need information about the number of instances of concept and role names. An estimate for this information can be computed in a preprocessing step by considering given data descriptions [49], or could be obtained by examining the result set of previously answered queries.

In Section 5.7 we have discussed that it is advantageous to compute and maintain an index $associated\_inds$ that allows us to find "obvious" instances by exploiting precompletion information. The index $associated\_inds$ is organized in such a way that retrieving the instances of a concept A, or one of its ancestors, requires (almost) constant time (in combination with the descendants, see Algorithm 12). This kind of index is particularly useful to provide bindings for variables if, despite all optimization attempts for deriving query execution plans, concept names must be used as generators. In addition, the index is used to estimate the cardinality of concept extensions. The estimates are used to compute an order relation for query atoms. The less the cardinality of a concept or a set of role fillers is assumed to be, the more priority is given to the query atom. Optimizing LUBM query $Q9$ with the techniques discussed above yields the following query execution plan.

$$\begin{aligned} Q9' : ans(x, y, z) \leftarrow\ & Faculty(y), teacherOf(y, z), Course(z), \\ & advisor^{-}(y, x), Student(x), takesCourse(x, z) \end{aligned}$$

Using this kind of rewriting, queries can be answered much more efficiently.

If the Tbox contains only GCIs of the form $\mathsf{A} \sqsubseteq \mathsf{A}_1 \sqcap \ldots \sqcap \mathsf{A}_n$, i.e., if the Tbox forms a hierarchy, the index-based retrieval discussed in Section 5.7 is complete (see [6]). However, this is not the case for LUBM. In LUBM, besides domain and range restrictions, axioms are also of the form $\mathsf{A} \equiv \mathsf{A}_1 \sqcap \mathsf{A}_2 \sqcap \ldots \sqcap \mathsf{A}_k \sqcap \exists \mathsf{R}_1.\mathsf{B}_1 \sqcap \ldots \sqcap \exists \mathsf{R}_m.\mathsf{B}_m$ (actually, $\mathsf{m} = 1$). If sufficient conditions with existential restrictions are specified as in the case of $Chair$, optimization is much more complex. In LUBM data descriptions, no individual is explicitly declared as a $Chair$ and, therefore, reasoning is required, which is known to be rather costly. If $Chair$ is used as a generator and not as a tester such as in the simple query $ans(x) \leftarrow Chair(x)$, optimization is even more important. The idea to optimize instance retrieval is to detect an additional number of obvious instances by transforming sufficient conditions into conjunctive queries.

6.2 Transforming Sufficient Conditions into Conjunctive Queries

Up to now we can detect obvious instances based on told and taxonomical information (almost constant time, see the previous section) as well as information extracted from the precompletion (linear time w.r.t. the number of remaining candidate individuals and a very fast test, see Section 5.4). Known non-instances can be determined with model merging techniques applied to individual pseudo models (also a linear process w.r.t. the number of remaining candidate individuals but with a very fast test, see Section 5.3). However, there might still be some candidates left. Using the results presented in Section 5 it is possible to use dependency-directed instance retrieval and binary partitioning. Our findings suggest that in the case of LUBM, for example for the concept $Chair$, the remaining refutational tableau proofs are very fast. However, for $Chair$ quite many candidates remain since there are many $Persons$ in LUBM. In application scenarios such as those we investigate with LUBM, we have 200 000 individuals and more (see the evaluation in Section 7), among them many $Persons$. Even if each single instance test lasts only a few dozen microseconds, query answering will be too slow, and hence additional techniques should be applied to solve the data description scalability problem.

Algorithm 16 $\mathit{rewrite}(\mathit{tbox}, \mathit{concept}, \mathit{var})$:

```
if $\mathit{unabsorbed\_gcis}(\mathit{tbox}) \neq \emptyset \vee \mathit{definition}(\mathit{concept}) = \top$ then
    return $\mathit{make\_atom}(\mathit{concept}, \mathit{var})$
else
    $\{\mathit{atom}_1, \ldots, \mathit{atom}_n\} := \mathit{rewrite\_0}(\mathit{tbox}, \mathit{concept}, \mathit{var}, \{\})$
      return $(\mathit{atom}_1, \ldots, \mathit{atom}_n)$
```

The central insight for another optimization technique is that conjunctive queries can be optimized according to the above-mentioned arguments whereas for concept-based queries, optimization is much harder to achieve. Let us consider the query $\mathit{ans}(x) \leftarrow \mathit{Chair}(x)$. For the concept Chair, a sufficient condition $\mathsf{Person} \sqcap \exists \mathsf{headOf}.\mathsf{Department}$ is given as part of the Tbox. Thus, in principle, we are looking for instances of $\mathsf{Person} \sqcap \exists \mathsf{headOf}.\mathsf{Department}$. The key to optimizing query answering becomes apparent if we transform the sufficient condition of Chair into a conjunctive query and derive the optimized version $Q15'$:

$$Q15 : \mathit{ans}(x) \leftarrow \mathit{Person}(x), \mathit{headOf}(x, y), \mathit{Department}(y)$$
$$Q15' : \mathit{ans}(x) \leftarrow \mathit{Department}(y), \mathit{headOf}^{-}(y, x), \mathit{Person}(x)$$

Because there exist fewer $\mathit{Departments}$ than $\mathit{Persons}$ in LUBM, the search for bindings for x is substantially more focused in $Q15'$ (which is the result of automatic query optimization, see above). In addition, in LUBM, the extension of $\mathit{Department}$ can be determined with simple index-based tests (only hierarchies are involved).

In addition, in the Tbox there is a domain restriction Professor for the role headOf, which can be exploited to further optimize the query by making atoms as specific as possible. Due to the domain restriction for headOf, the variable x in Q15' must refer to a Professor instance, which can be made explicit. If we further exploit that Professor is subsumed by Person, it is clear that the atom $Person(x)$ can be dropped.

$$Q15'' : \mathit{ans}(x) \leftarrow \mathit{Department}(y), \mathit{headOf}^{-}(y, x), \mathit{Professor}(x)$$

With the Chair example one can easily see that the standard approach for instance retrieval can be optimized dramatically with rewriting concept query atoms if certain conditions are met.

The idea of the rewriting is to implement the inverse of the contraction or rolling-up technique (see [22] and also Section 5.1). Here, however, existential restrictions are "rolled-down" to conjunctive queries. The rewriting is reminiscent of a rewriting introduced by Borgida in [9]. In this work, description logic concepts are translated to first-order logic formulae. The transformation approach discussed in this section is also reminiscent to a transformation approach discussed in [45]. Note, however, that in our approach, we additionally consider domain and range restrictions for roles in order to maximize information that can be used for exploiting indexes.

The rewriting algorithm is defined in Algorithms 16, 17, and 18. Every concept query atom $C(x)$ used in a conjunctive query is replaced with $\mathit{rewrite}(\mathit{query\_tbox}, C, x)$ (and afterwards, the query is optimized with the techniques describe above). If the transformation approach is applied to $Q15$, the query $Q15''$ is derived.

Some auxiliary functions are used. The function $\mathit{definition}(\mathsf{C})$ returns sufficient conditions for a concept name C (the result is a concept), the function $\mathit{make\_atom}(c, v)$ returns

Algorithm 17 $\mathit{rewrite\_0}(\mathit{tbox}, \mathit{concept}, \mathit{var}, \mathit{exp})$:

```
if $\mathit{definition}(\mathit{concept}) = \top \vee \mathit{concept} \in \mathit{exp}$ then
    return $\{\mathit{make\_atom}(\mathit{concept}, \mathit{var})\}$
else
    ;; catch installs a marker to which the control flow can be thrown
    catch $\mathit{not\_rewritable}$
        $\mathit{rewrite\_1}(\mathit{tbox}, \mathit{concept}, \mathit{definition}(\mathit{tbox}, \mathit{concept}), \mathit{var}, \{\mathit{concept}\} \cup \mathit{exp})$
```

Algorithm 18 $\mathit{rewrite\_1}(\mathit{tbox}, \mathit{concept\_name}, \mathit{definition}, \mathit{var}, \mathit{exp})$:

```
if ($\mathit{definition} = \mathsf{A}$) where A is a concept name then
    return $\mathit{rewrite\_0}(\mathit{tbox}, \mathit{definition}, \mathit{var}, \{\mathit{definition}\} \cup \mathit{exp})$
else
    if ($\mathit{definition} = \exists \mathsf{R.C}$) then
        $\mathit{filler\_var} := \mathit{fresh\_variable}()$
        return $\{R(\mathit{var}, \mathit{filler\_var})\} \cup \mathit{rewrite\_0}(\mathit{tbox}, \mathsf{C}, \mathit{filler\_var}, \mathit{exp})$
               $\cup\ \mathit{rewrite\_0}(\mathit{tbox}, \mathit{role\_domain}(R), \mathit{var}, \mathit{exp})$
               $\cup\ \mathit{rewrite\_0}(\mathit{tbox}, \mathit{role\_range}(R), \mathit{filler\_var}, \mathit{exp})$
    else
        if ($\mathit{definition} = \mathsf{C_1} \sqcap \ldots \sqcap \mathsf{C_n}$) then
            return $\mathit{rewrite\_1}(\mathit{tbox}, \mathit{concept\_name}, \mathsf{C_1}, \mathit{var}, \mathit{exp})$
                   $\cup \cdots \cup$
                   $\mathit{rewrite\_1}(\mathit{tbox}, \mathit{concept\_name}, \mathsf{C_n}, \mathit{var}, \mathit{exp})$
        else
            ;; throw the control flow out of rewrite_1 recursion
            ;; back to the call to rewrite_1 in rewrite_0 and
            ;; return {concept_name(var)}
            throw $\mathit{not\_rewritable}\ \{\mathit{concept\_name}(\mathit{var})\}$
```

the atom $c(v)$, and the function $\mathit{unabsorbed\_gcis}(\mathit{tbox})$ indicates whether there are some unabsorbed GCIs left after GCI transformation, i.e., $\mathcal{T}_G \neq \emptyset$ (see Section 3.3, the result is a set of concepts). In addition, we use a function $\mathit{fresh\_variable}$ that generates a new variable that was not used before. The functions $\mathit{role\_domain}$ and $\mathit{role\_range}$ return the domain and range restrictions of a role (after GCI absorption).

If there is no specific definition or the function $\mathit{unabsorbed\_gcis}$ returns a non-empty set, rewriting is not applied (see Algorithm 16). It is easy to see that the rewriting approach is sound. However, it is complete only under specific conditions, which can be automatically detected. If we consider the Tbox $\mathcal{T} = \{\mathsf{D} \equiv \exists \mathsf{R.C}\}$, the Abox $\mathcal{A} = \{\mathsf{i} : \exists \mathsf{R.C}\}$ and the query $\mathit{ans}(x) \leftarrow D(x)$, then due to the algorithm presented above the query will be rewritten as $\mathit{ans}(x) \leftarrow R(x, y), C(y)$. For variable bindings, the query language nRQL (see above) considers only those individuals that are explicitly mentioned in the Abox. Thus i is not part of the result set because there is no binding for y in the Abox $\mathcal{A}$. Examining the LUBM Tbox and Abox it becomes clear that in this case for every $\exists \mathsf{R.C}$ which is applicable to an individual i there already exist assertions $(\mathsf{i}, \mathsf{j}) : \mathsf{R}$ and $\mathsf{j} : \mathsf{C}$ in the original Abox (LUBM's design was inspired by a database schema). Existential restrictions are fulfilled by named individuals (see Section 5.10). However, even if this is not the case, the technique can be employed under some circumstances. Usually, in order to construct a model (or a completion to be more precise), tableau provers create a new individual for each constraint of the form $\mathsf{i} : \exists \mathsf{R.C}$ and add corresponding concept and role assertions (if not already present). These newly created individuals are called anonymous individuals. Let us assume, during the initial Abox consistency test a completion is found. As we have discussed above, a precompletion is computed by removing all constraints that depend on a choice point. If there is no such constraint, the precompletion is identical to the completion that the tableau prover computed. Then, the set of bindings for variables generated by $\mathit{fresh\_variable}()$ used in Algorithm 18 is extended to the anonymous individuals found in the precompletion. The rewriting technique for concept query atoms is applicable (i.e., is complete) under these conditions. Even if the rewriting technique is not complete (because something has been removed from a completion to derive a precompletion), it can be employed to reduce the set of candidates for binary partitioning techniques that can speed up this process considerably in the average case. In the following section, we will evaluate how the optimization techniques introduced up to now provide a contribution to the data description scalability problem.

7 Evaluation

In this section we discuss the impact of the optimization techniques investigated in this article by considering runtimes for the traditional inference service of Abox realization and for KB specific queries. The runtimes we present in this section are used to demonstrate the order of magnitude of time resources that are required for solving inference problems when the complexity of the input problem is increased. They allow us to analyze the impact of the presented optimization techniques.

7.1 Benchmark Knowledge Bases

We selected two sets of knowledge bases for benchmarking. The first set consists of ten KBs with large Aboxes that were derived from various applications of DL technology within the semantic web community. The second set contains three knowledge bases with very large Aboxes.

7.1.1 A Selection of Large Application Knowledge Bases

The OWL knowledge bases discussed in this section contain relatively simple Tboxes but large Aboxes. The characteristics of these KBs are summarized in Table 1 and 2. The second column in Table 1 characterizes the Tbox logic determined by RACER. The Tbox logic of the input file might be different because RACER might add disjunctions to the processed KB due to GCI absorption (e.g., this is the case for the LUBM KB). The third column shows the number of concept names, the fourth indicates the number of roles, and the fifth presents the number of Tbox axioms. The second column in Table 2 gives the Abox logic determined by RACER. The third to fifth columns show the number of individuals, individual assertions, and role assertions. It is interesting to note that the Abox logic is sometimes slightly more complex than the Tbox logic.

The Tbox/Abox logic is indicated using the standard DL terminology (see [4] for details). We additionally denote the DL supporting only conjunction and primitive negation by $\mathcal{L}$ and $\mathcal{L}^-$ stands for $\mathcal{L}$ without primitive negation. Both variants of $\mathcal{L}$ also admit simple concept inclusions whose left-hand sides consist only of a name. The notation "$(\mathcal{D})$" is used to denote the use of concrete domain expressiveness (see [33,4] for details). The occurrence of "$(\mathcal{D})$" is caused by OWL datatype properties that are restricted by number restrictions (and RACER applies concrete domain reasoning to these constructs) whereas "$(\mathcal{D}^-)$" denotes "$(\mathcal{D})$" without number restricted datatype properties. Furthermore, the use of functional roles is denoted as f and of transitive roles as R^+ (or $\mathcal{S}$).

Table 1 Tbox characteristics of the 10 application KBs used for benchmarking.

Knowledge Base	Tbox Logic	Concept Names	Roles	Axioms
SoftEng	$\mathcal{L}^{-}\mathcal{H}$	37	30	76
SEMINTEC	$\mathcal{FL}_0 f$	59	24	345
FungalWeb	$\mathcal{ALCH}(\mathcal{D})$	3 601	77	7 209
InfoGlue	$\mathcal{ALCH}$	41	37	83
WebMin 1	$\mathcal{ALEH}f(\mathcal{D}^{-})$	444	175	1 714
WebMin 2	$\mathcal{ALEH}f(\mathcal{D}^{-})$	520	204	1 893
LUBM	$\mathcal{ALCH}$	43	41	85
FERMI	$\mathcal{L}$	5 136	15	10 265
UOBM-lite	$\mathcal{ALCH}f$	51	49	101
VICODI	$\mathcal{L}^{-}\mathcal{H}$	194	10	387

Table 2 Abox characteristics of the 10 application KBs used for benchmarking.

Knowledge Base	Abox Logic	Inds	Ind. Assertions	Role Assertions
SoftEng	$\mathcal{L}^{-}\mathcal{H}R^{+}$	6 735	5 595	25 896
SEMINTEC	$\mathcal{FL}_0 f$	17 941	17 941	41 174
FungalWeb	$\mathcal{ALCH}(\mathcal{D})$	12 556	12 705	1159
InfoGlue	$\mathcal{SH}$	15 464	15 937	88 316
WebMin 1	$\mathcal{ALCH}f(\mathcal{D}^{-})$	1 427	9 193	1 146
WebMin 2	$\mathcal{ALCH}f(\mathcal{D}^{-})$	6 532	28 676	15 915
LUBM	$\mathcal{SH}(\mathcal{D}^{-})$	17 174	51 207	49 336
FERMI	$\mathcal{EL}$	700	9 998	650
UOBM-lite	$\mathcal{SH}f(\mathcal{D}^{-})$	5 674	10 790	11 970
VICODI	$\mathcal{L}^{-}\mathcal{H}$	16 942	16 942	36 704

It is also important to mention that RACER computes the Tbox/Abox logic of a KB by analyzing it in detail. For instance, if a KB declares a transitive (inverse) role (as in the case of LUBM) but never uses this role within an axiom or, in the case of an inverse role, there does not exist an interaction between a role and its inverse (e.g., $\exists \mathsf{R} . \forall \mathsf{R}^{-} . \mathsf{C}$), then the Tbox logic does not refer to transitive or inverse roles. The same methodology is applied to Aboxes but the logic of the Abox is always at least as expressive as the one of its Tbox.

The selected ten knowledge bases can be described as follows.

- The SoftEng ontology was created by a reverse engineering approach [55] where an abstract representation of Java code is represented as a KB and DL inference services were used to reason about security concerns.
- The SEMINTEC[3] ontology models financial services.
- The FungalWeb[4] ontology is an outcome of an project using DL technology in the context of fungal genomics [48,5].
- The InfoGlue ontology is the by-product of a DL-based approach to support the comprehension and maintenance of software systems [47].
- The two Web Mining ontologies (WebMin 1 and 2) are proprietary and were contributed by users of RACER.
- The LUBM[5] ontology [27–29] (see also below) represents the structural organization of a set of universities (with a varying number of departments). Although LUBM is

[3] `http://www.cs.put.poznan.pl/alawrynowicz/semintec.htm`
[4] `http://www.cs.concordia.ca/FungalWeb/`
[5] `http://swat.cse.lehigh.edu/projects/lubm/index.htm`

an artificial benchmark the distribution of entities matches real-world universities quite well.
- The FERMI[6] ontology was generated in the context of a project about formalization and experimentation on the retrieval of multimedia information.
- The UOBM-lite ontology [42] was derived from LUBM but has a more complicated Tbox and Abox structure. It represents the structural organization of one university (with all departments).
- The VICODI[7] ontology is about European history.

7.1.2 Very Large Knowledge Bases

In order to better investigate Abox scalability, the following three knowledge bases with very large Aboxes are used.

- The Lehigh University Benchmark (LUBM, [27–29]) is used with two different Tboxes (lite and normal) and 6 different Abox sizes ranging from 5 to 50 universities (with all departments). The original LUBM Tbox is in $\mathcal{ELH}$ but RACER's GCI absorption process adds disjunctions to the axioms resulting in a Tbox which is in $\mathcal{ALCH}$. In the case of LUBM another absorption is possible that avoids the addition of disjunctions but this is currently not supported by RACER. However, a slight modification to the original LUBM Tbox can avoid the unnecessary addition of disjunctions. Thus, we decided to investigate two variants of LUBM, the original one, called LUBM, and the modified one, called LUBM-lite. The characteristics of the LUBM KBs are described in Table 9.
- The UOBM or UOB ontology [42] was derived from LUBM. For benchmarking the variant based on OWL-lite is used with 5 different Abox sizes ranging from 1 to 5 universities (with all departments). The UOBM ontology extends the LUBM ontology and adds more expressiveness to both Tbox and Abox. The characteristics of the UOBM KBs are shown in Figure 10.
- The Wordnet knowledge base (version 1.7.1)[8] is an OWL-DL KB representing the WordNet 1.7.1 lexical database [18]. The $\mathcal{L}^-$ Tbox of Wordnet contains 84K concept names and 85K axioms. Its $\mathcal{ELR}^+(\mathcal{D}^-)$ Abox consists of 269K individuals, 548K individual and 304K role assertions (see also Figure 5 and Table 8).

Besides having very large Aboxes the LUBM and UOB knowledge bases were selected because they are well-known and have been used for various evaluations of DL reasoners. They have the disadvantage of being synthetic which is offset by their advantage of being scalable. Moreover, the authors of these ontologies argue that their knowledge bases reflect typical data that could have been extracted from databases and they underwent substantial efforts in making them very close to application data [29,42]. The Wordnet ontology was selected because it also has a very large Abox and its database counterpart [18] is used in numerous applications worldwide.

7.2 Evaluation of Large Knowledge Bases

As outlined earlier we evaluated the application KBs using the inference service of Abox realization which heavily relies on the techniques introduced in the previous sections. This

[6] `http://www.dcs.gla.ac.uk/fermi/`
[7] `http://www.vicodi.org/`
[8] `http://taurus.unine.ch/files/wordnet171.owl.gz`

Table 3 Processing times of application KBs using the standard optimization setting (in secs).

Knowledge Base	Load	Classification	Realization
SoftEng	7.43	0.03	5.61
SEMINTEC	7.25	0.07	18.60
FungalWeb	3.20	1.71	59.20
InfoGlue	28.00	0.04	161.00
WebMin 1.00	1.57	0.90	14.80
WebMin 2.00	8.63	0.74	104.00
LUBM	14.20	0.04	46.40
FERMI	7.56	3.35	1.33
UOBM-lite	4.73	0.05	61.60
VICODI	5.65	0.07	34.10

(Load = load time, Tbox = Tbox classification time, Abox = Abox realization time)

kind of Abox indexing w.r.t. to concept names stress-tests these techniques and is especially suitable if a sufficient number of specific benchmark queries for the selected ontologies are not available. Another argument for realization are RDF query languages such as SPARQL, which heavily relies on concept names for querying. In cases where a reasonable number of queries were available (as in the case of LUBM and UOBM) we additionally tested these KBs with these sets of queries (as detailed in Section 7.3).

7.2.1 Experimental Settings

All experiments were conducted by switching on or off selected optimization techniques (as introduced in the previous sections) in order to assess the positive (and sometimes also negative) impact of these techniques on the runtimes. The tests were conducted on a Sun server V890 with 8 dual core processors and 64 GB of main memory (although all these tests usually require less than 2 GB of memory usage and each test was executed on a single processor). For each setting the average runtimes of the 10 application KBs were sequentially computed where each KB test was repeated 5 times. Each setting was tested with a fresh image of RACER, i.e., the restarts of RACER correspond to the number of settings tested, and the given runtimes include the time for garbage collection. The runtimes using the "standard" optimization setting of RACER are presented in Table 3. The second column shows the time for loading the KB, the third for classification, and the fourth for realization (including the time for the initial Abox consistency test). It can be seen that the classification times can be mostly neglected (as expected) and the realization times vary between less than 2 seconds and almost 3 minutes. The average runtimes are usually inflated by the overhead caused by garbage collection.

For the evaluation of the application KBs the following 11 parameters were used to switch optimization techniques on or off.

P1 Individual pseudo model merging (see Section 3.1): switched on by default.
P2 Abox contraction (see Section 5.1): switched on by default.
P3 Sets-of-individuals-at-a-time instance retrieval (see Section 5.7): switched on by default. If it is switched off, linear instance retrieval (see Section 5.2) is selected.
P4 Abox completion (see Section 5.3): switched on by default.
P5 Abox precompletion (see Section 5.4): switched on by default.
P6 Binary instance retrieval (see Section 5.5): switched on by default.
P7 Dependency-based instance retrieval (see Section 5.6): switched on by default.

P8 Static index-based instance retrieval (see Section 5.7): switched off by default (time and memory demands can be excessive in general).
P9 Dynamic index-based retrieval (see Section 5.8): switched off by default (time and memory demands can be excessive in general).
P10 OWL-DL datatype properties simplification (see Section 5.9): switched on by default.
P11 Re-use of role assertions for existential restrictions (see Section 5.10): switched on by default.

To verify the effectiveness of the optimization techniques related to these 11 parameters, 17 different benchmark settings were created. These settings were selected to demonstrate the positive (and sometimes also negative) effect of optimization techniques. In most settings exactly one optimization is switched off (on) that is normally switched on (off) in the standard setting. In a few settings up to three optimizations are switched off (on) because they either depend on one another or one would compensate for others that are disabled. The exact composition of these 17 settings is shown in in the rows of Table 4. The settings have the following meaning:

1. Standard setting with only static (P8) and dynamic index-based retrieval (P9) disabled. All the following settings are based on this standard setting.
2. Abox completion (P4) switched off.
3. Abox precompletion (P5) switched off.
4. Abox completion (P4) and precompletion (P5) switched off.
5. Individual pseudo model merging (P1) switched off.
6. Abox contraction (P2) switched off.
7. Sets-of-individuals-at-a-time (P3) switched off.
8. Sets-of-individuals-at-a-time (P3) and binary instance retrieval (P6) switched off.
9. Sets-of-individuals-at-a-time (P3), binary instance retrieval (P6), and dependency-based instance retrieval (P7) switched off.
10. OWL-DL datatype simplification (P10) switched off.
11. Re-use of role assertions (P11) switched off.
12. Static index-based instance retrieval (P8) switched on.
13. Dynamic index-based instance retrieval (P9) switched on.
14. Dependency-based instance retrieval (P7) switched off.
15. Binary instance retrieval (P6) switched off.
16. Abox precompletion (P5) and dependency-based instance retrieval (P7) switched off.
17. Abox precompletion (P5) and binary instance retrieval (P6) switched off.

7.2.2 Evaluation Using the Realization Inference Service

The first series of evaluations were performed with the KBs introduced in Section 7.1.1 and under the conditions described at the beginning of Section 7.2.1. For all KBs the direct types for all individuals mentioned in the associated Abox were computed and verified. Each test was performed with all 17 settings described above. Each Abox realization was repeated five times and the average of these five runs is shown in the Tables 5 and 6. In the following we analyze the results by focusing on (i) the best and worst settings per KB, and (ii) for each setting the KB having the most positive and negative impact. The graphs in Figure 3 and 4 illustrate these results per KB.

The SoftEng KB is only affected by S5 which disables individual pseudo model merging and results in an increase of runtime by a factor of 100.

Table 4 Composition of the selected 17 different optimization settings.

Setting	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
1	·	·	·	·	·	·	·	×	×	·	·
2	·	·	·	×	·	·	·	×	×	·	·
3	·	·	·	·	×	·	·	×	×	·	·
4	·	·	·	×	×	·	·	×	×	·	·
5	×	·	·	·	·	·	·	×	×	·	·
6	·	×	·	·	·	·	·	×	×	·	·
7	·	·	×	·	·	·	·	×	×	·	·
8	·	·	×	·	·	×	·	×	×	·	·
9	·	·	×	·	·	×	×	×	×	·	·
10	·	·	·	·	·	·	·	×	×	×	·
11	·	·	·	·	·	·	·	×	×	·	×
12	·	·	·	·	·	·	·	·	×	·	·
13	·	·	·	·	·	·	·	×	·	·	·
14	·	·	·	·	·	·	×	×	×	·	·
15	·	·	·	·	·	×	·	×	×	·	·
16	·	·	·	·	×	·	×	×	×	·	·
17	·	·	·	·	×	×	·	×	×	·	·

Setting 1 is the standard setting, · used for switched on, × for switched off.

Table 5 Abox realization with optimization settings 1-8 (in seconds).

Knowledge Base	S1	S2	S3	S4	S5	S6	S7	S8
SoftEng	5.6	5.4	5.4	5.5	480	5.4	6.0	5.7
SEMINTEC	18.6	18.9	17.6	18.9	TO	18.8	12.2	11.5
FungalWeb	59.2	66.6	59.1	58.9	461	59.0	64.5	53.8
InfoGlue	160	113	TO	TO	TO	161	TO	TO
WebMin 1	14.8	14.9	14.5	15.0	45.4	14.8	13.2	13.5
WebMin 2	104	57.7	215	215	TO	57.8	124	67.9
LUBM	46.4	49.9	133	136	TO	58.8	185	189
FERMI	1.33	1.33	1.3	1.37	6.90	1.31	1.19	1.28
UOBM-lite	61.6	68.4	914	935	834	61.6	TO	TO
VICODI	34.1	22.3	23.5	23.0	TO	33.0	21.8	26.0

TO = timeout (after 1000 seconds),
S1 = standard, S2 = completion off, S3 = precompletion off,
S4 = completion+precompletion off, S5 = ind. pseudo model merging off,
S6 = contraction off, S7 = sets-of-inds-at-a-time off,
S8 = sets-of-inds-at-a-time+binary instance retrieval off.

A similar observation holds for the SEMINTEC KB, which timed out after 1000 seconds. Its best runtimes are for S7-S9, which switch the sets-of-individuals-at-a-time technique off. This indicates a 50% overhead for this technique.

The FungalWeb KB has its runtime increased by a factor of 8 for setting S5. Otherwise it remains mostly unaffected if 10% variations are ignored.

The InfoGlue KB's best setting is S15, which switches off binary instance retrieval. This indicates that the partitioning scheme only causes overhead in this case. The second-best one is S2 (no completion). This indicates that the completion tests are mostly unsatisfiable and thus wasted due to the incompleteness of this technique. InfoGlue timed out for S3-S5, S7-S9, and S16-S17. The size of this KB and its Tbox/Abox logics ($\mathcal{ALCHI}/\mathcal{SH}$) explain the timeout for S3-S4, which switch off the precompletion technique and cause the overhead

Table 6 Abox realization with optimization settings 9-17 (in seconds).

Knowledge Base	S9	S10	S11	S12	S13	S14	S15	S16	S17
SoftEng	5.67	5.42	5.45	5.47	5.45	5.6	5.5	6.45	6.23
SEMINTEC	11.6	18.3	16.2	18.9	18.8	18.1	18.0	18.9	22.9
FungalWeb	53.2	59.4	56.7	59.8	59.5	60.2	57.8	65.3	59.5
InfoGlue	TO	133	158	162	159	207	96.0	TO	TO
WebMin 1	13.5	22.3	14.8	14.8	14.8	15.2	14.4	14.9	14.5
WebMin 2	68.0	140	55.2	57.7	88.3	78.1	64.3	202	200
LUBM	191	68.7	66.6	59.3	58.7	59.1	40.6	170	200
FERMI	1.19	1.27	1.2	1.3	1.32	1.34	1.36	1.27	1.26
UOBM-lite	TO	66.8	66.2	61.5	61.8	55.9	58.1	TO	915
VICODI	25.7	33.1	21.0	33.2	32.9	23.7	28.6	29.8	23.4

TO = timeout (after 1000 seconds),
S9 = sets-of-inds-at-a-time+binary+dependency-based instance retrieval off,
S10 = datatype simplification off, S11 = Re-use of role assertions off,
S12 = static index-based instance retrieval on,
S13 = dynamic index-based instance retrieval on,
S14 = dependency-based instance retrieval off, S15 = binary instance retrieval off,
S16 = precompletion+dependency-based instance retrieval off,
S17 = precompletion+binary instance retrieval off.

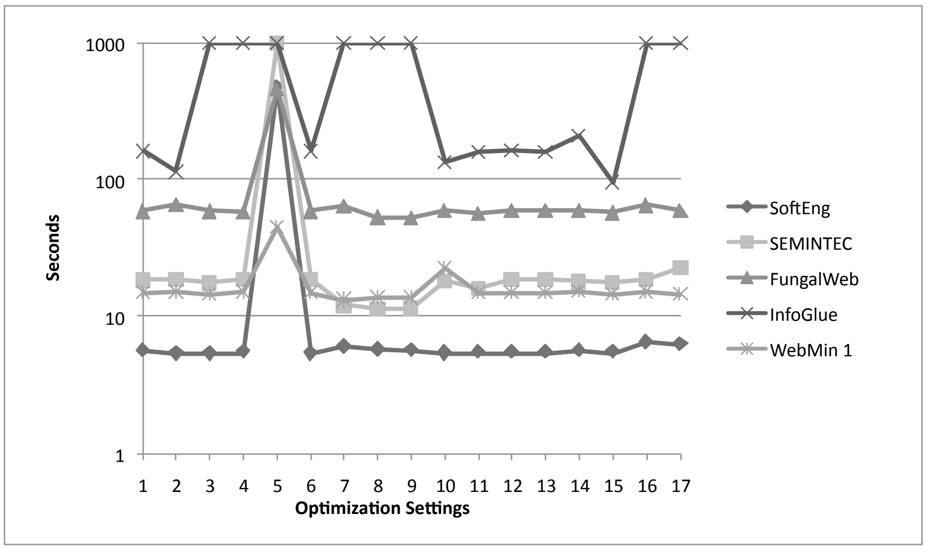


Fig. 3 Abox realization graphs of the first 5 application KBs.

of recomputation of assertions making up the precompletion. A similar effect as for SEMINTEC also occurs for InfoGlue for S5. The disabled sets-of-individuals-at-a-time technique explains the timeout for S7-S9 because the then enabled linear instance retrieval causes too much overhead.

The WebMin 1 KB remains mostly unaffected except by S5 (3 times slower) and S10 (50% slower), which disables the datatype simplification.

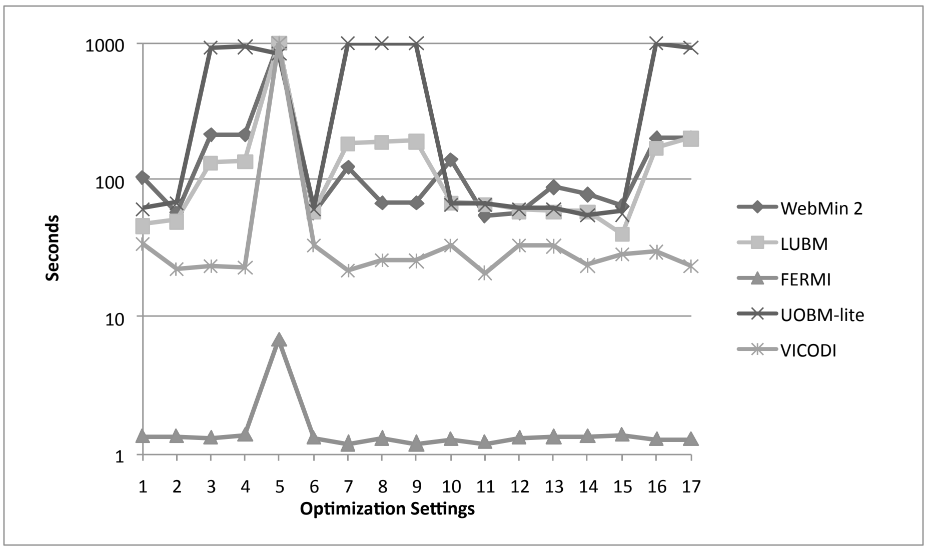


Fig. 4 Abox realization graphs of the last 5 application KBs.

The best settings for the WebMin 2 KB are S2, S6, S11, where the overhead of these techniques is saved, and S12, where the static index-based instance retrieval compensates for the overhead of the other techniques. WebMin 2's worst settings are S3-S4, which switch the precompletion off.

Due to its size LUBM timed out for S5 and its best runtime is for S15, which is 10% faster than the standard one. A factor of 4 in the increase of the runtime can be noticed for S7-S9 due to the overhead of linear instance retrieval.

The observations for the FERMI KB are similar to SoftEng, although S5 only causes an increase of a factor of 5. This is due to the very simple structure of this KB.

For UOMB-lite the best is S14, which disabled dependency-based instance retrieval. S3-S4 cause an increase of a factor of 15 because they switch off the precompletion. A similar observation can be made for S5. Both can be explained by the size of the KB. S7-S9 timed out because sets-of-individuals-at-a-time was disabled and linear instance retrieval enabled.

The VICODI KB timed out for setting S5 and shows a variation of up to 50% in the other settings.

The standard setting (S1) was selected with the goal to ensure a good overall performance. This is generally confirmed by these benchmarks. For S2 some KBs (e.g., WebMin2) have smaller runtimes than in S1. The positive effect can be explained by the low success rate of the completion test (e.g., 0% for WebMin 2, 66% for InfoGlue) and the incompleteness of this technique in case it reported an (possibly unavoidable) inconsistency. For some KBs such as LUBM and UOBM-lite we notice a slowdown of 10%. S3 shows that the Abox precompletion technique is advantageous for most KBs and even essential for InfoGlue and UOBM-lite. This is due to the reduced overhead in rebuilding initial data structures. S4 indicates that the missing precompletion dominates the increase in runtime. S5 has a very detrimental effect on the runtime. This clearly demonstrates the effectiveness of the individual pseudo model merging technique for Abox realization. S6 is virtually identical to the standard setting except for WebMin 2, where we observe a speed-up of almost 50%. These results indicates that this technique does not seem to be very effective for these

KBs. S7 mostly shows the positive effect of the sets-of-individuals-at-a-time technique. It is essential for InfoGlue and UOBM-lite, which both timed out. LUBM is 3 times slower but VICODI is 30% faster. S8-S9 demonstrate that the disabled sets-of-individuals-at-a-time technique dominates the slowdown. The results for S10 are mixed. Some KBs such as InfoGlue show a performance gain due to reduced overhead while others such as WebMin 1/2 have an increased runtime. S11 shows a similar pattern where WebMin 2 is twice as fast as in the standard setting but others slowed down. S12 and S13 are different from the previous ones because they switch techniques *on* that are disabled by default. The only exception for S12 is WebMin 2, which doubled in speed. All others are in the range of the standard setting. The next section discusses scenarios where these 2 settings are very favorable. S13 behaves similarly to S12 but WebMin 2 has only a speed-up of 20%. S14 shows also mixed results. InfoGlue slowed down by 25% while VICODI and WebMin 2 increased in speed. So, dependency-based instance retrieval is sometimes favorable because it helps to separate "clash culprits" and sometimes it causes unnecessary overhead. S15 shows clearly that binary instance retrieval does not improve the runtimes for realization of the 10 KBs. S16-S17 need to be compared to S3, which also switches off precompletion. LUBM slowed down by 30-50% and UOBM-lite timed out for S16. In these cases S16 and S17 have a positive effect due to the disabled precompletion. All other results are similar to S3.

7.2.3 Evaluation of Static and Dynamic Index-based Retrieval

In the following we illustrate the effectiveness of static and dynamic index-based instance retrieval (see Algorithms 12 and 15) with the help of the LUBM KB (1 university) used in the previous section. We used 2 concepts for querying their instances. The first one is the predefined concept Chair, the second one is a concept conjunction describing a Chair whose email address (specified by the datatype property emailAddress) has to be equal to the string `"FullProfessor2@Department12.University0.edu"`.

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 following query set ('...' stands for the string`"FullProfessor2@Department12.University0.edu"` and the DL notation $(= \mathsf{emailAddress}\ \ldots)$ is used to restrict the value of the datatype property emailAddress).

$$\{\mathit{instance\_retrieval}(\mathsf{Chair}, \mathit{LUBM}),\\ \mathit{instance\_retrieval}(\mathsf{Chair} \sqcap (= \mathsf{emailAddress}\ \ldots), \mathit{LUBM})\}$$

Two different strategies were used: (i) *Strategy 1*: all instances of the concept Chair are retrieved first; (ii) *Strategy 2*: all instances of the concept $\mathsf{Chair} \sqcap (= \mathsf{emailAddress}\ \ldots)$ are retrieved first.

The query set was evaluated with 4 different settings: (i) standard (S1), (ii) static index-based instance retrieval switched on (S12), (iii) dynamic index-based instance retrieval switched on (S13), (iv) sets-of-individuals-at-a-time switched off and static index-based instance retrieval switched on (S7+S12). The runtimes of the query set – 4 settings and each under the 2 strategies – are shown in Table 7. The columns list (from left to right) the selected strategy, selected optimization settings, time for Abox consistency test, time for Abox realization, time to compute the instances of Chair, time to compute the instances of the concept conjunction, total benchmark time.

Table 7 Runtimes (in secs) of LUBM (1 university) instance retrieval (2 strategies).

Strategy	Setting	Consistency	Realization	Chair	*Anon*	Total
1	S1	17.9	–	12.0	585	649
2	S1	18.0	–	1.38	601	611
1	S12	18.1	15.2	0.001	4.02	66
2	S12	18.1	15.0	0.001	3.97	65
1	S13	10.0	–	6.66	4.42	61
2	S13	15.6	–	1.75	554	605
1	S7+S12	18.3	129	0.001	1.07	174
2	S7+S12	18.1	131	0.001	1.04	175

S1 = standard, S7 = sets-of-inds-at-a-time off,
S12 = static index-based instance retrieval on,
S13 = dynamic index-based instance retrieval on.
Anon stands for $\mathsf{Chair} \sqcap (= \mathsf{emailAddress}\ \ldots)$

Table 8 Characteristics of the Wordnet 1.7.1 knowledge base.

Tbox Logic	CN	R	Axioms	Abox Logic	Inds	Ind. Ass.	Role Ass.
$\mathcal{L}^-$	84 609	40	85 664	$\mathcal{ELR}^+(\mathcal{D}^-)$	269 684	548 578	304 362

(CN = no. of concept names, R = no. of roles)

Table 7 reveals that the query with the concept conjunction is expensive in the standard setting (which does not keep an index or cache about previous query results) and the query execution order has only a minor impact. Both strategies give a speed-up of 2 orders of magnitude with setting S12 for the concept conjunction query. In this case, the time for Abox realization paid off well. The runtimes are insensitive to the query execution orders. The next setting (S13) clearly demonstrates that dynamic index-based retrieval is advantageous to the standard setting provided the proper execution order of the queries (based on query concept subsumption) has been chosen. One might conclude that static index-based retrieval is always preferable to the dynamic one but in case of larger KBs the initial overhead for index building might increase significantly. The last setting (S7+S12) reveals the efficiency of the sets-of-individuals-at-a-time technique (see Section 5.7.2) in contrast to linear instance retrieval. Due to S7 the overhead to setup the static index increased by almost 1 order of magnitude.

7.3 Evaluation of Very Large Knowledge Bases

The previous section evaluated the presented optimization techniques for instance retrieval mostly on the basis of Abox realization. In this section the evaluation of instance retrieval is continued with very large knowledge bases. The first very large KB is Wordnet, which is evaluated with Abox realization only due to lack of a sufficient number of specific benchmark queries. The other two very large KBs are LUBM and UOBM. They are evaluated using the execution of grounded conjunctive queries, which were designed by the developers of these KBs. Both KBs are tested for Abox size scalability using the standard optimization setting. Furthermore, they are also evaluated against the 17 settings from the previous section.

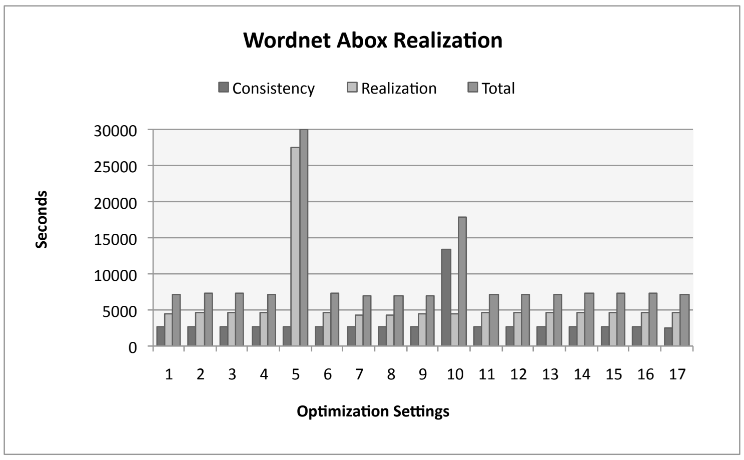


Fig. 5 Runtimes for Wordnet Abox realization using the 17 opt. settings.

7.3.1 Wordnet

The Wordnet OWL-DL KB consists of three files with a total size of 102MB. Its characteristics are shown in Table 8. The $\mathcal{L}^-$ Tbox consists of 84K concept names and 85K axioms defining a given taxonomy. The Abox logic is $\mathcal{ELR}^+(\mathcal{D}^-)$ and indicates the use of transitive roles and OWL-DL datatype properties. The Tbox load time is 130 seconds, and its classification time is 90.44 seconds. By analogy to the previous section we evaluated Abox realization using the 17 settings. The results are displayed in Figure 5. The runtimes for most settings are in the range for 8 000 seconds and do not vary much. Setting S5 (ind. pseudo model merging off) timed out after 30 000 seconds. This result is in line with the lessons learnt from testing large KBs. S10 (datatype simplification off) is the other exception with a runtime increased by a factor of 2.5. This clearly demonstrates the advantage of the datatype property optimization technique. S15-S17 show a speedup of roughly 10% due to the reduced overhead of the disabled techniques.

7.3.2 LUBM

The LUBM benchmark has the big advantage of being scalable. LUBM was tested with 5-50 university, each with all departments. This results for 50 universities in 1082K individuals, 3355K individual assertions, and 3298K role assertions. Two different Tboxes were used in order to investigate the influence of the GCI absorption technique (see also the discussion in Section 7.1.2). An overview about the characteristics and sizes of the LUBM benchmarks is given in Table 9. The Tbox logic of LUBM-lite is $\mathcal{ELH}$, its Abox logic is $\mathcal{ELHR}^+(\mathcal{D}^-)$. The Tbox logic of LUBM is $\mathcal{ALCH}$, its Abox logic is $\mathcal{SH}(\mathcal{D}^-)$.

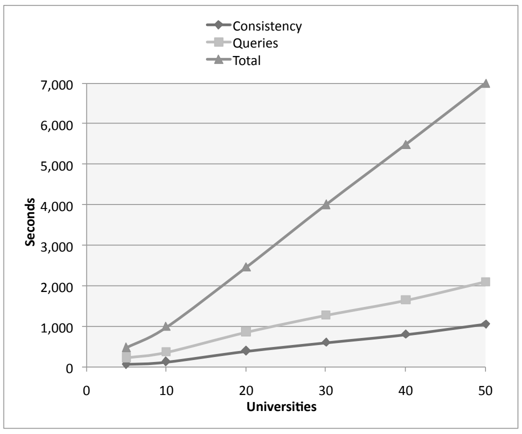
Each benchmark was evaluated with 14 grounded conjunctive queries designed by the authors of LUBM. The benchmark log recorded the runtime for the following phases: (i) loading the input files; (ii) data structure setup for the KB. These runtimes are identical for both Tbox variants. The other recorded runtimes are (see the second to fifth columns in Figures 6 and 7): (iii) time for the initial Abox consistency test that precedes the first query execution and initializes appropriate data structures and indexes; (iv) nRQL Abox index generation time; (v) time to execute all 14 queries; (vi) total time consumed by the

Table 9 LUBM/LUBM-lite Abox characteristics (time in seconds).

U	Individuals	Ind. Assertions	Role Assertions	Load	Prep
5	102 368	315 139	309 393	90	60
10	207 426	641 822	630 753	196	153
20	437 555	1 356 017	1 332 029	472	456
30	645 954	2 001 556	1 967 308	726	946
40	864 222	2 676 802	2 630 656	990	1 417
50	1 082 818	3 355 749	3 298 813	1 296	1 758

U = no. of universities, Load = load time, Prep = KB preparation time.

U	C	I	Q	T
5	67	39	228	478
10	128	153	363	982
20	391	307	857	2 460
30	600	487	1 274	4 003
40	798	669	1 642	5 487
50	1 055	859	2 100	7 000



U = no. of universities, C = time for initial Abox consistency test,
I = nRQL Abox index generation time, Q = nRQL query execution time,
T = total benchmark time.

Fig. 6 LUBM-lite query runtimes (in seconds).

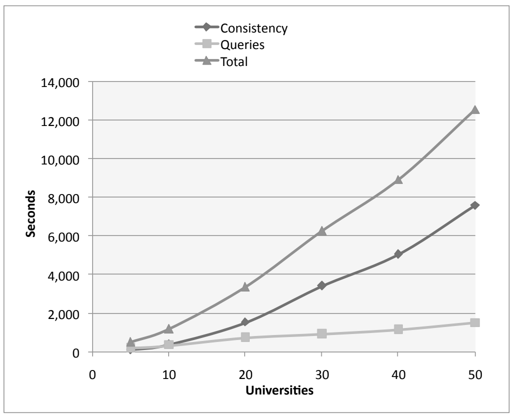
benchmark. The right part of Figures 6 and 7 shows a graph displaying curves for the runtime of the Abox consistency test (dashed line), query execution (dotted line), and the total benchmark time (solid line).

The graph for LUBM-lite in Figure 6 gives evidence of RACER's excellent scalability for this benchmark type. The total runtime is even dominated by the load and preparation time while Abox consistency and query execution apparently exhibit a straight line with a much smaller gradient than that of the total runtime.

The gradients of the apparently straight lines shown in the graph in Figure 7 are similar for query execution but steeper for the Abox consistency test and the total runtime. The more complex Tbox causes no penalty for query execution. On the contrary, the query execution is even faster by roughly 30%. The Abox consistency test requires now more than 50% of the total runtime and dominates the benchmark results. This can be explained by the treatment of disjunctions in axioms that were added by the GCI absorption.

We also conducted a second study with LUBM (using both Tbox variants) and a selected Abox size of 10 universities. The 14 queries were executed using the 17 settings from the previous section. The Figures 8 and 9 show in the left part the recorded runtimes and in the right part a bar chart illustrating the results (using dark grey for consistency, middle grey for queries, light grey for total runtime). Please note the use of the logarithmic scale in the bar charts.

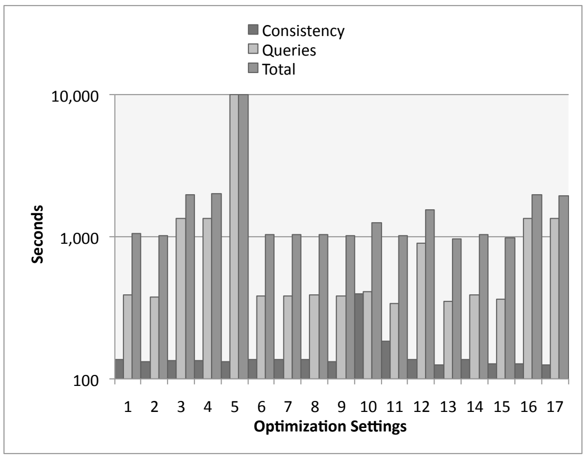
U	C	I	Q	T
5	118	38	208	510
10	412	81	351	1 183
20	1 542	158	746	3 359
30	3 422	255	934	6 259
40	5 058	326	1 156	8 915
50	7 579	427	1 512	12 541



U = no. of universities, C = time for initial Abox consistency test,
I = nRQL Abox generation time, Q = nRQL query execution time,
T = total benchmark time.

Fig. 7 LUBM query runtimes (in seconds).

S	C	Q	T
1	135	387	1 039
2	132	371	1 003
3	133	1 326	1 962
4	133	1 340	1 979
5	132	TO	TO
6	135	381	1 026
7	136	379	1 033
8	135	385	1 034
9	131	377	1 007
10	391	410	1 247
11	184	339	1 002
12	135	890	1 541
13	125	348	955
14	136	386	1 035
15	127	360	971
16	126	1 331	1 941
17	125	1 322	1 927

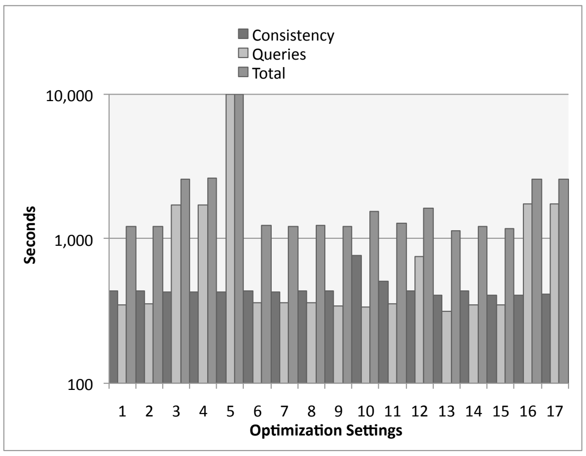


TO = timeout (after 10 000 seconds),
S = selected optimization setting, C = time for initial Abox consistency test,
Q = nRQL query execution time, T = total benchmark time.

Fig. 8 LUBM-lite (10 universities) query runtimes (in seconds).

The obtained results for LUBM-lite (see Figure 8) demonstrate that the Abox consistency test is mostly unaffected. Its runtime tripled for setting S10 (datatype simplification switched off) and increased by 30% for S11 (re-use of role assertion switched off). The execution of the queries timed out for S5 (individual pseudo model merging switched off), although Abox realization has not been performed. This emphasizes the importance of this technique also for query-based instance retrieval. The query runtime tripled for S3-S4 and S16-S17, which switch precompletion off. It indicates the effectiveness of the precompletion technique. The other notable slowdown occurred for S12 (factor of 2), which switches on

S	C	Q	T
1	433	345	1 210
2	431	353	1 216
3	429	1 691	2 556
4	430	1 704	2 598
5	427	TO	TO
6	435	357	1 229
7	427	360	1 216
8	434	359	1 224
9	431	343	1 205
10	769	337	1 537
11	504	356	1 284
12	432	744	1 610
13	408	315	1 138
14	433	348	1 214
15	406	349	1 168
16	408	1 736	2 556
17	410	1 730	2 556



TO = timeout (after 10 000 seconds),
S = selected optimization setting, C = time for initial Abox consistency test,
Q = nRQL query execution time, T = total benchmark time.

Fig. 9 LUBM (10 universities) query runtimes (in seconds).

the static index-based instance retrieval. It is obvious that the overhead to build and maintain the index is too high and does not pay off for the execution of the queries.

The runtimes for the Abox consistency test for LUBM (see Figure 9) have tripled compared to LUBM-lite as expected due to the added disjunctions in the transformed axioms. The recorded runtimes are rather uniform. Setting S10 shows an almost doubled runtime and S11 a slight increase. The efficiency of these techniques is compensated by the increased overhead for dealing with disjunctions. Query execution timed out again for S5. Setting S3-S4 and S16-S17, which switch off precompletion, are now a factor 4-5 slower than the standard setting. By analogy to LUBM-lite S12 demonstrates an increased overhead (factor of 2).

7.3.3 UOBM

The third and last very large KB discussed in this section is the UOBM-lite benchmark. It is also scalable and was tested with 1-5 universities, each with all departments. The characteristics of the KB and the benchmarks are shown in Figure 10. The Tbox of UOBM-lite consists of 51 concept names, 49 role names, and 101 axioms, and its logic is $\mathcal{ALCf}$ (determined after GCI absorption). The Abox adds datatype properties and its logic is $\mathcal{ALCf}(\mathcal{D}^-)$. The size of the benchmark for 5 universities is 138K individuals, 509K individual assertions, and 563K role assertions.

Each benchmark was evaluated with 15 grounded conjunctive queries designed by the authors of UOBM. The benchmark has the same structure as for LUBM. The runtimes given in Figure 10 show that RACER's Abox consistency performance scales well for up to 3 universities. The runtime increased by a factor of 2 for 4 universities and a factor of 7 for 5 universities. This degradation of performance is caused by inefficiencies in managing internal data structures and is subject to further investigations.

U	Inds	Ind. Ass.	Role Ass.	L	P	Cons	I	Q	T
1	43 642	116 092	129 695	35	16	100	15	446	608
2	66 900	200 018	222 492	61	35	353	13	1 571	2 041
3	85 055	272 663	302 425	84	59	482	28	3 272	3 920
4	109 919	378 956	419 364	115	111	1 096	31	13 791	15 132
5	138 452	509 902	563 699	160	197	7 670	40	TO	TO

TO = timeout (after 30 000 seconds),
U = no. of universities, L = load time, P = KB preparation time,
Cons = time for initial Abox consistency test, I = query index generation time,
Q = nRQL query execution time, T = total benchmark time.

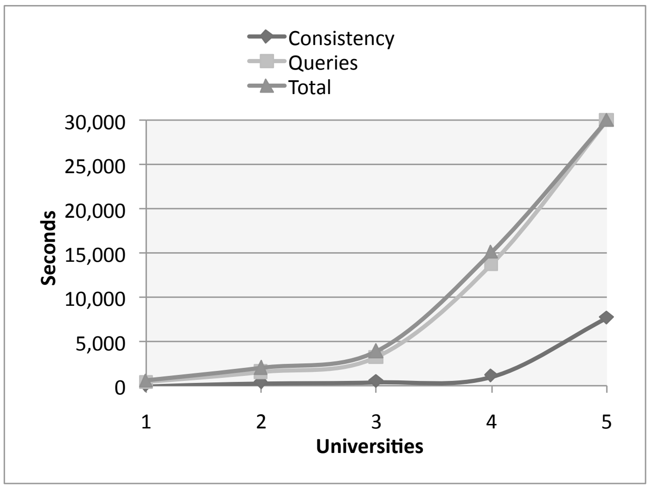


Fig. 10 UOBM-lite benchmark characteristics and runtimes (time in seconds).

In contrast to LUBM the UOBM benchmark does not make (and does not allow to impose) the unique name assumption. The query execution time also scales well for up to 3 universities. However, for 4 universities it increased by a factor 4 and timed out for 5 universities after 30 000 seconds. The graph in the lower part of Figure 10 displays the curves for the Abox consistency test (dark gray line), query execution (light gray line), and the total benchmark time (medium gray line). The non-linear trend can be easily noticed. It is interesting to remark that 99.86% of the query runtime is spent for 3 of the 15 queries. This performance asks for a refinement of existing or the design of new optimization techniques.

For these reasons the second study conducted with UOBM was restricted to a size of 3 universities. We tested the 15 queries using the 17 settings. The results are displayed in Figure 11 where the left bar chart uses a linear and the right one a logarithmic scale (using dark grey for consistency, middle grey for queries, light grey for total runtime). Setting S12, which switches static index-based instance retrieval on, timed out after 30 000 seconds. This result clearly demonstrates that in the case of these 15 queries Abox realization is not worth the effort. S5 switches individual pseudo model merging off and caused an increase of runtime by a factor of 5. Again, this gives evidence for the effectiveness of this technique for instance retrieval without realization. By analogy to LUBM one can notice a slight increase for S3-S4 and S16-S17, which switch off the precompletion, and S10, which disables

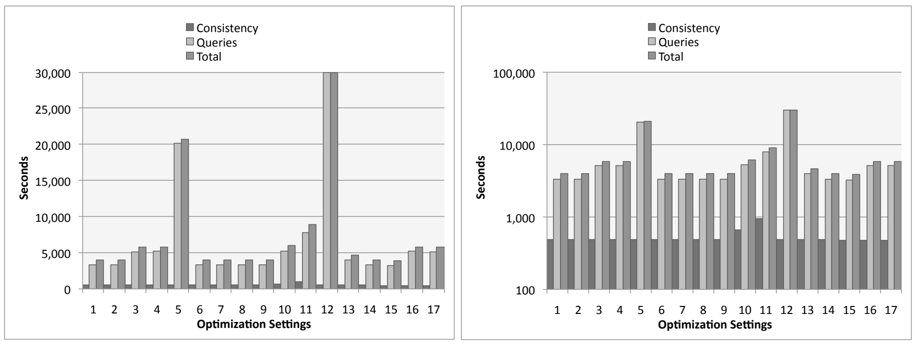


Fig. 11 UOBM-lite (3 universities) query runtimes (in seconds, timeout after 30 000 seconds).

datatype property simplification. S11 doubled the runtime due to the disabled re-use of role assertions.

8 Conclusion and Future Work

In this article we demonstrated optimization techniques that make Abox inferences based on tableau-based DL systems suitable for many non-naive applications. We motivated the techniques described in this article with the semantic web scenario and its web ontology language OWL. 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-DL. Nevertheless, they already demonstrate the need for more advanced optimization techniques.

With a set of 10 selected large application KBs, which contain large Aboxes, we could demonstrate that the described and implemented optimization techniques are effective for these types of KBs. However, keeping the balance between creating data structures for faster query answering and relying on memory-conserving algorithms remains still a challenge and is subject to ongoing research. For instance, the index-based instance retrieval techniques can increase memory demand significantly, and often the overhead to maintain these data structures can easily dominate the runtime.

Concerning the 10 large KBs one could see that various optimization settings have positive or negative impacts on the runtime. This even depends on the usage profile and the requested inference services. It turned out that the standard setting seems to be a good compromise between overhead for indexing and potential speedup. The automatic selection of an optimal optimization setting for submitted KBs is an ongoing research topic and will be investigated in future research.

We take Wordnet, LUBM and UOBM as representatives for very large and mostly deterministic data descriptions (encoded as Aboxes) that can be found in practical applications. The investigations reveal that description logic systems can be optimized to also be able to deal with large bulks of assertional knowledge quite effectively. LUBM indicates that performance scales well with an increasing number of assertions given the expressivity of the language used in the ontology meets certain requirements. Our work is based on a tableau calculus which has shown to be reliable if expressivity is increased for some parts (see the results for full LUBM and UOBM). The linear shape of the curves for LUBM suggests that the proposed technology ensures that performance scales if high expressivity is not required.

LUBM is in a sense too inexpressive but the benchmark allows us to study the Abox scalability problem. Note that optimizations, for instance, for qualified number restrictions are not known for other approaches to query answering. The results for UOBM indicate that scalability could not be fully achieved in all cases. This is also a topic for future research.

Note that we argue that the optimization techniques investigated in this paper are advantageous not only for RACER but also for other tableau-based systems. Future work will investigate more optimizations for large Aboxes and more expressive Tboxes.

Concerning binary instance retrieval it is an open issue whether specific heuristics for grouping individuals into partitions can be provided. This is ongoing work. For investigating dependency-based instance retrieval, in this work, a dependency-tracking infrastructure w.r.t. all completions has been developed (see Section 5.6). In future work these techniques will also be used for providing explanations for instance retrieval and for other reasoning problems.

Acknowledgements We thank the developers of the LUBM and UOBM benchmarks and also the developers of the other knowledge bases for making their work available to the research community. We also express our gratitude to Michael Wessel for his research on nRQL, and to the anonymous reviewers for their helpful comments. The work presented in this article has been partially funded by the EU projects TONES (FP6-7603) and BOEMIE (FP6-027538).

References

1. A. Acciarri, D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, and R. Rosati. Quonto: Querying ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 1670–1671, 2005.
2. F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.J. Profitlich. 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, 1994.
3. F. Baader, I. Horrocks, and U. Sattler. Description logics as ontology languages for the semantic web. In D. Hutter and W. Stephan, editors, *Mechanizing Mathematical Reasoning: Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday*, volume 2605 of *Lecture Notes in Artificial Intelligence*, pages 228–248. Springer-Verlag, 2005.
4. Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
5. C. Baker, A. Shaban-Nejad, X. Su, V. Haarslev, and G. Butler. Semantic web infrastructure for fungal enzyme biotechnologists. *Journal of Web Semantics*, 4(3):168–180, 2006.
6. Sean Bechhofer, Ian Horrocks, and Daniele Turi. The OWL instance store: System description. In *Proc. of the 20th Int. Conf. on Automated Deduction (CADE-20)*, Lecture Notes in Artificial Intelligence, pages 177–181. Springer, 2005.
7. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American, May 2001*, may 2001.
8. A. Borgida and R. Brachman. Loading data into description reasoners. *ACM SIGMOD Record*, 22(2):217–226, 1993.
9. Alexander Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1–2):353–367, 1996.
10. Paolo Bresciani. Querying databases from description logics. In *Proceedings of Knowledge Representation Meets Databases (KRDB'95), Saarbrücken, Germany, DFKI-Research-Report D-95-12*, pages 1–4, 1995.
11. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In *Proc. of the 2005 Description Logic Workshop (DL 2005)*, pages 49–60. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/, 2005.
12. D. Calvanese, G. De Giacomo, and M. Lenzerini. Decidability of query containment under constraints. In *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 149–158. ACM Press, 1998.

13. A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Nineth ACM Symposium on Theory of Computing*, pages 77–90, 1977.
14. Yu Ding and Volker Haarslev. Towards efficient reasoning for description logics with inverse roles. In *Proceedings of the 2005 International Workshop on Description Logics (DL-2005), Edinburgh, Scotland, UK, July 26-28*, pages 208–215, 2005.
15. Yu Ding and Volker Haarslev. Tableau caching for description logics with inverse and transitive roles. In *Proceedings of the 2006 International Workshop on Description Logics (DL-2006), Lake District, UK, May 30 - June 1*, pages 143–149, 2006.
16. J. Dolby, A. Fokoue, A. Kalyanpur, A. Kershenbaum, L. Ma, E. Schonberg, and K. Srinivas. Scalable semantic retrieval through summarization and refinement. In *21st Conference on Artificial Intelligence (AAAI)*, pages 299–304, 2007.
17. Francesco M. Donini and Fabio Massacci. Exptime tableaux for ALC. *Artifical Intelligence*, 124(1):87–138, 2000.
18. C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.
19. Richard Fikes, Patrick Hayes, and Ian Horrocks. OWL-QL—a language for deductive query answering on the Semantic Web. *J. of Web Semantics*, 2(1):19–29, 2004.
20. A. Fokoue, A. Kershenbaum, L. Ma, E. Schonberg, and K. Srinivas. The summary abox: Cutting ontologies down to size. In *Proc. of International Semantic Web Conference (ISWC)*, pages 343–356, 2006.
21. H. Garcia-Molina, J.D. Ullman, and J. Widom. *Database Systems: The Complete Boook*. Prentice Hall, 2092.
22. B. Glimm and I. Horrocks. Query answering systems in the semantic web. In *CEUR workshop proceedings of KI-2004 Workshop on Applications of Description Logics (ADL-04)*, 2004.
23. Birte Glimm, Ian Horrocks, Carsten Lutz, and Uli Sattler. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, pages 399–404, 2007.
24. R. Goré, A. Leitsch, and T. Nipkow, editors. *Proceedings of the International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, 2001, Siena, Italy*, Lecture Notes in Computer Science. Springer-Verlag, June 2001.
25. Rajeev P. Goré and Linh Nguyen. Exptime tableaux with global caching for description logics with transitive roles, inverse roles and role hierarchies. In *Proceedings of TABLEAUX'2007, International Conference, Automated Reasoning with Analytic Tableaux and Related Methods, Aix en Provence, France, 3-6 July*, volume 4548 of *LNAI*, pages 133–148. Springer-Verlag, 2007.
26. Y. Guo and J. Heflin. A scalable approach for partitioning owl knowledge bases. In *Proc. of the 2nd International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2006), Athens, Georgia, USA*, pages 47–60, 2006.
27. Y. Guo, J. Heflin, and Z. Pan. Benchmarking DAML+OIL repositories. In *Proc. of the Second Int. Semantic Web Conf. (ISWC 2003)*, number 2870 in LNCS, pages 613–627. Springer Verlag, 2003.
28. Y. Guo, Z. Pan, and J. Heflin. An evaluation of knowledge base systems for large OWL datasets. In *Proc. of the Third Int. Semantic Web Conf. (ISWC 2004)*, volume 3298 of *LNCS*, pages 274–288. Springer Verlag, 2004.
29. Y. Guo, Z. Pan, and J. Heflin. Lubm: A benchmark for owl knowledge base systems. *Journal of Web Semantics*, 3(2):158–182, 2005.
30. V. Haarslev and R. Möller. An empirical evaluation of optimization strategies for ABox reasoning in expressive description logics. In P. Lambrix et al., editor, *Proceedings of the International Workshop on Description Logics (DL'99), July 30 - August 1, 1999, Linköping, Sweden*, pages 115–119, June 1999.
31. V. Haarslev and R. Möller. RACER system description. In Goré et al. [24], pages 701–705.
32. V. Haarslev and R. Möller. Optimization techniques for retrieving resources described in OWL/RDF documents: First results. In *Ninth International Conference on the Principles of Knowledge Representation and Reasoning, KR 2004, Whistler, BC, Canada, June 2-5*, pages 163–173, 2004.
33. V. Haarslev, R. Möller, and M. Wessel. The description logic $\mathcal{ALCNH}_{R^+}$ extended with concrete domains: A practically motivated approach. In Goré et al. [24], pages 29–44.
34. Volker Haarslev and Ralf Möller. Consistency testing: The RACE experience. In *Proceedings International Conference Tableaux'2000*, volume 1847 of *Lecture Notes in Artificial Intelligence*, pages 57–61. Springer-Verlag, 2000.
35. Volker Haarslev, Ralf Möller, and Anni-Yasmin Turhan. 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*, pages 61–75. Springer-Verlag, 2001.
36. I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

37. I. Horrocks, U. Sattler, and S. Tobies. Reasoning with individuals for the description logic $\mathcal{SHIQ}$. In David MacAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction (CADE-17)*, Lecture Notes in Computer Science, pages 482–496, Germany, 2000. Springer Verlag.
38. I. Horrocks and S. Tessaris. A conjunctive query language for description logic Aboxes. In *Proc. of the 17th Nat. Conf. on Artificial Intelligence (AAAI 2000)*, pages 399–404, 2000.
39. I. Horrocks and S. Tobies. Reasoning with axioms: Theory and practice. In A.G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proceedings of Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR'2000), Breckenridge, Colorado, USA, April 11-15, 2000*, pages 285–296, April 2000.
40. Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible $\mathcal{SROIQ}$. In *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 57–67. AAAI Press, 2006.
41. Ian Horrocks and Sergio Tessaris. Querying the semantic web: a formal approach. In Ian Horrocks and James Hendler, editors, *Proc. of the 13th Int. Semantic Web Conf. (ISWC 2002)*, number 2342 in Lecture Notes in Computer Science, pages 177–191. Springer-Verlag, 2002.
42. L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, and S. Liu. Towards a complete owl ontology benchmark. In *Proc. of 3rd European Semantic Web Conference (ESWC)*, pages 124–139, 2006.
43. Ralf Möller, Volker Haarslev, and Michael Wessel. On the scalability of description logic instance retrieval. In *Proceedings of the 29th Annual German Conference on Artificial Intelligence, June 14-19, Bremen, Germany, LNCS, Springer Verlag*, Lecture Notes in Artificial Intelligence, pages 171–184. Springer Verlag, 2006.
44. B. Motik and U. Sattler. A comparison of reasoning techniques for querying large description logic aboxes. In *Proceedings of the 13th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2006), Phnom Penh, Cambodia, November 13-17*, volume 4246 of *LNCS*, pages 227–241. Springer, 2006.
45. B. Motik, R. Volz, and A. Maedche. Optimizing query answering in description logics using disjunctive deductive databases. In *Proceedings of the 10th International Workshop on Knowledge Representation Meets Databases (KRDB-2003)*, pages 39–50, 2003.
46. Bernhard Nebel. *Reasoning and Revision in Hybrid Representation Systems*, volume 422 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1990.
47. J. Rilling, Y. Zhang, V. Haarslev, W. Meng, and R. Witte. A unified ontology-based process model for software maintenance and comprehension. In *Proceedings of the ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS/UML 2006), T. Kühne (Ed.), LNCS 4364, Springer-Verlag*, pages 56–65, 2007.
48. A. Shaban-Nejad, C. Baker, V. Haarslev, and G. Butler. The fungalweb ontology: Semantic web challenges in bioinformatics and genomics. In *Semantic Web Challenge - Proceedings of the 4th International Semantic Web Conference, Nov. 6-10, Galway, Ireland, Springer-Verlag, LNCS, Vol. 3729*, pages 1063–1066, 2005.
49. Evren Sirin and Bijan Parsia. Optimizations for answering conjunctive abox queries: First results. In *Proceedings of the 2006 International Workshop on Description Logics (DL-2006), Lake District, UK, May 30 - June 1*, pages 215–222, 2006.
50. F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL web ontology language reference, http://www.w3.org/tr/owl-guide/, 2003.
51. T. Weithöner, T. Liebig, M. Luther, S. Böhm, F.W. von Henke, and O. Noppens. Real-world reasoning with owl. In *Proceedings of 4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, June 3-7, 2007, LNCS 4519*, pages 296–310. Springer-Verlag, 2007.
52. Timo Weithöner, Thorsten Liebig, and Günther Specht. Storing and Querying Ontologies in Logic Databases. In *Proceedings of The first International Workshop on Semantic Web and Databases (SWDB'03)*, pages 329 – 348, Berlin, Germay, September 2003.
53. M. Wessel and R. Möller. A high performance semantic web query answering engine. In *Proc. of the 2005 Description Logic Workshop (DL 2005)*, pages 84–95. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/, 2005.
54. M. Wessel and R. Möller. Flexible software architectures for ontology-based information systems. *Journal of Applied Logic (Special Issue on Empirically Successful Systems)*, 2008.
55. Y. Zhang, J. Rilling, and V. Haarslev. An ontology based approach to software comprehension - reasoning about security concerns in source code. In *Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC 2006), IEEE Computer Society Press*, pages 333–342, 2006.
56. Z. Zhang. Ontology query languages for the semantic web: A performance evaluation. Master's thesis, University of Georgia, 2005.

57. M. Zuo and V. Haarslev. High performance absorption algorithms for terminological reasoning. In *Proceedings of the 2006 International Workshop on Description Logics (DL-2006), Lake District, UK, May 30 - June 1*, pages 159–166, 2006.