A Comparison of Reasoning Techniques for Querying Large Description Logic ABoxes

Boris Motik and Ulrike Sattler

University of Manchester
Manchester, UK

Abstract. Many modern applications of description logics (DLs) require answering queries over large data quantities, structured according to relatively simple ontologies. For such applications, we conjectured that reusing ideas of deductive databases might improve scalability of DL systems. Hence, in our previous work, we developed an algorithm for reducing a DL knowledge base to a disjunctive datalog program. To test our conjecture, we implemented our algorithm in a new DL reasoner KAON2, which we describe in this paper. Furthermore, we created a comprehensive test suite and used it to conduct a performance evaluation. Our results show that, on knowledge bases with large ABoxes but with simple TBoxes, our technique indeed shows good performance; in contrast, on knowledge bases with large and complex TBoxes, existing techniques still perform better. This allowed us to gain important insights into strengths and weaknesses of both approaches.

1 Introduction

Description logics (DLs) are a family of knowledge representation formalisms with applications in numerous areas of computer science. They have long been used in information integration [1, Chapter 16], and they provide a logical foundation for OWL—a standardized language for ontology modeling in the Semantic Web [10]. A DL knowledge base is typically partitioned into a terminological (or schema) part, called a *TBox*, and an assertional (or data) part, called an *ABox*. Whereas some applications rely on reasoning over large TBoxes, many DL applications involve answering queries over knowledge bases with small and simple TBoxes, but with large ABoxes. For example, the documents in the Semantic Web are likely to be annotated using simple ontologies; however, the number of annotations is likely to be large. Similarly, the data sources in an information integration system can often be described using simple schemata; however, the data contained in the sources is usually very large.

Reasoning with large data sets was extensively studied in the field of deductive databases, resulting in several techniques that have proven themselves in practice. Motivated by the prospect of applying these techniques to query answering in description logics, in our previous work we described a novel reasoning algorithm [12] that reduces a $\mathcal{SHIQ}$ knowledge base KB to a disjunctive datalog program $\mathsf{DD}(KB)$ while preserving the set of relevant consequences. This

algorithm is quite different from tableau algorithms [1, Chapter 2] and their optimizations [9], used in state-of-the-art DL reasoners such as RACER [8], FaCT++ [22], or Pellet [17].

We conjectured that our algorithm will scale well to knowledge bases with large ABoxes and simple TBoxes. In particular, we expected great benefits from techniques such as magic sets [4] or join-order optimizations. Furthermore, we identified a Horn fragment of $\mathcal{SHIQ}$ [13], for which our algorithm exhibits polynomial data complexity (that is, the complexity measured in the size of the ABox, assuming the TBox is fixed in size).

To test our conjecture, we implemented the reduction algorithm in a new DL reasoner KAON2.[1] To obtain an efficient system, we developed several optimizations of the initial algorithm and of known implementation techniques. In this paper we outline the design of the system and overview the employed techniques. Due to lack of space, we are unable to present all optimizations in full detail; for more information, please refer to [15].

Providing an objective account of the performance of our approach proved to be difficult because there are no widely recognized benchmarks for query answering. To fill this gap, we created a benchmark suite consisting of several ontologies with TBoxes of varying size and complexity, and with large ABoxes. In this paper, we discuss the guidelines we followed in selecting the test data. Our benchmarks are freely available on the Web,[2] and we hope that they can provide a starting point for a standard DL test suite.

Finally, we conducted extensive performance tests with KAON2, RACER, and Pellet. To obtain a complete picture of the performance of our algorithms, apart from ABox reasoning tests, we also performed several TBox reasoning tests. The results were twofold, and were roughly in line with our expectations. Namely, on ontologies with a small TBox but a large ABox, our algorithm outperformed its tableau counterparts; however, on ontologies with a complex TBox but a small ABox, existing algorithms exhibited superior performance. We discuss these results, and provide insight into strengths and weaknesses of either algorithm. This may provide useful guidance to developers of future DL systems.

Summing up, our reasoning algorithm provides good performance for knowledge bases which do not rely too heavily on modal reasoning, but are more akin to logic programs. However, the boundary between the two extreme use-cases is not clear-cut. As a consequence, we now have a more comprehensive set of reasoning techniques for expressive DLs, allowing the users to choose the one that best suits the needs of their application.

2 Preliminaries

We now present the syntax and the semantics of the DL $\mathcal{SHIQ}$ [11]—the formalism underlying KAON2. Given a set of role names N_R, a $\mathcal{SHIQ}$ *role* is either some $R \in N_R$ or an *inverse role* R^- for $R \in N_R$. A $\mathcal{SHIQ}$ *RBox* $KB_{\mathcal{R}}$ is a

[1] http://kaon2.semanticweb.org/

[2] http://kaon2.semanticweb.org/download/test_ontologies.zip

Table 1. Semantics of $\mathcal{SHIQ}$ by Mapping to FOL

Translating Concepts to FOL	
$\pi_y(A, X) = A(X)$	$\pi_y(C \sqcap D, X) = \pi_y(C, X) \wedge \pi_y(D, X)$
$\pi_y(\neg C, X) = \neg \pi_y(C, X)$	$\pi_y(\forall R.C, X) = \forall y : R(X, y) \rightarrow \pi_x(C, y)$
$\pi_y(\geq n\, S.C, X) = \exists y_1, \ldots, y_n : \bigwedge S(X, y_i) \wedge \bigwedge \pi_x(C, y_i) \wedge \bigwedge y_i \not\approx y_j$	
Translating Axioms to FOL	
$\pi(C \sqsubseteq D) = \forall x : \pi_y(C, x) \rightarrow \pi_y(D, x)$	$\pi(C(a)) = \pi_y(C, a)$
$\pi(R \sqsubseteq S) = \forall x, y : R(x, y) \rightarrow S(x, y)$	$\pi(R(a, b)) = R(a, b)$
	$\pi(a \circ b) = a \circ b$ for $\circ \in \{\approx, \not\approx\}$
$\pi(\mathsf{Trans}(R)) = \forall x, y, z : R(x, y) \wedge R(y, z) \rightarrow R(x, z)$	
Translating KB to FOL	
$\pi(R) = \forall x, y : R(x, y) \leftrightarrow R^-(y, x)$	
$\pi(KB) = \bigwedge_{R \in N_R} \pi(R) \wedge \bigwedge_{\alpha \in KB_\mathcal{T} \cup KB_\mathcal{R} \cup KB_\mathcal{A}} \pi(\alpha)$	

X is a meta variable and is substituted by the actual variable. π_x is obtained from π_y by simultaneously substituting $x_{(i)}$ for all $y_{(i)}$, respectively, and π_y for π_x.

finite set of role inclusion axioms $R \sqsubseteq S$ and transitivity axioms $\mathsf{Trans}(R)$, for R and S $\mathcal{SHIQ}$ roles. For $R \in N_R$, we set $\mathsf{Inv}(R) = R^-$ and $\mathsf{Inv}(R^-) = R$, and assume that $R \sqsubseteq S \in KB_\mathcal{R}$ ($\mathsf{Trans}(R) \in KB_\mathcal{R}$) implies $\mathsf{Inv}(R) \sqsubseteq \mathsf{Inv}(S) \in KB_\mathcal{R}$ ($\mathsf{Trans}(\mathsf{Inv}(R)) \in KB_\mathcal{R}$). A role R is said to be *simple* if $\mathsf{Trans}(S) \notin KB_\mathcal{R}$ for each $S \sqsubseteq^* R$, where $\sqsubseteq^*$ is the reflexive-transitive closure of $\sqsubseteq$.

Given a set of *concept names* N_C, the set of $\mathcal{SHIQ}$ concepts is the minimal set such that each $A \in N_C$ is a $\mathcal{SHIQ}$ concept and, for C and D $\mathcal{SHIQ}$ concepts, R a role, S a simple role, and n a positive integer, $\neg C$, $C \sqcap D$, $\forall R.C$, and $\geq n\, S.C$ are also $\mathcal{SHIQ}$ concepts. We use $\top$, $\bot$, $C_1 \sqcup C_2$, $\exists R.C$, and $\leq n\, S.C$ as abbreviations for $A \sqcup \neg A$, $A \sqcap \neg A$, $\neg(\neg C_1 \sqcap \neg C_2)$, $\neg \forall R.\neg C$, and $\neg(\geq (n+1)\, S.C)$, respectively. A TBox $KB_\mathcal{T}$ is a finite set of concept inclusion axioms of the form $C \sqsubseteq D$. An ABox $KB_\mathcal{A}$ is a finite set of axioms $C(a)$, $R(a, b)$, and (in)equalities $a \approx b$ and $a \not\approx b$. A knowledge base KB is a triple $(KB_\mathcal{R}, KB_\mathcal{T}, KB_\mathcal{A})$. The semantics of KB is given by translating it into first-order logic by the operator π from Table 1.

A *query* Q over KB is a conjunction of literals $A(s)$ and $R(s, t)$, where s and t are variables or constants, R is a role, and A is an atomic concept. In our work, we assume that all variables in a query should be mapped to individuals explicitly introduced in the ABox. Then, a mapping θ of the free variables of Q to constants is an *answer* of Q over KB if $\pi(KB) \models Q\theta$.

3 KAON2 Architecture

KAON2 is a DL reasoner developed at the University of Manchester and the University of Karlsruhe. The system can handle $\mathcal{SHIQ}$ knowledge bases extended with *DL-safe* rules—first-order clauses syntactically restricted in a way that makes the clauses applicable only to individuals mentioned in the ABox, thus ensuring decidability. KAON2 implements the following reasoning tasks: deciding knowledge base and concept satisfiability, computing the subsumption hierarchy, and answering conjunctive queries without distinguished variables (i.e., all variables of a query can be bound only to explicit ABox individuals, and not

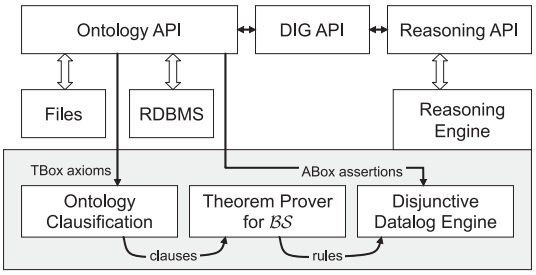


Fig. 1. KAON2 Architecture

to individuals introduced by existential quantification). It has been implemented in Java 1.5.

Figure 1 describes the technical architecture of KAON2. The *Ontology API* provides ontology manipulation services, such as adding and retrieving axioms. The API fully supports OWL and the Semantic Web Rule Language (SWRL) at the syntactic level. Several similar APIs already exist, such as the OWL API [3] or Jena.[3] However, to obtain an efficient system, we needed complete control over the internals of the API, and could thus not reuse an existing implementation. Ontologies can be saved in files, using either OWL RDF[4] or OWL XML[5] syntax. Alternatively, ABox assertions can be stored in a relational database (RDBMS): by mapping ontology entities to database tables, KAON2 will query the database on the fly during reasoning.

The *Reasoning API* allows one to invoke various reasoning tasks, and to retrieve their results.

All APIs can be be invoked either locally, using KAON2 as a dynamic library, or remotely, for example, through the DL Implementors Group (DIG) interface.

The central component of KAON2 is the *Reasoning Engine*, which is based on the algorithm for reducing a $\mathcal{SHIQ}$ knowledge base KB to a disjunctive datalog program $\mathsf{DD}(KB)$ [12]. To understand the intuition behind this algorithm, considering the knowledge base $KB = \{C \sqsubseteq \exists R.E_1,\ E_1 \sqsubseteq E_2,\ \exists R.E_2 \sqsubseteq D\}$. For an individual x in C, the first axiom implies existence of an R-successor y in E_1. By the second axiom, y is also in E_2. Hence, x has an R-successor y in E_2, so, by the third axiom, x is in D. The program $\mathsf{DD}(KB)$ contains the rules $E_2(x) \leftarrow E_1(x)$ and $D(x) \leftarrow R(x,y), E_2(x)$, corresponding to the second and the third axiom, respectively. However, the first axiom of KB is not represented in $\mathsf{DD}(KB)$; instead, $\mathsf{DD}(KB)$ contains the rule $D(x) \leftarrow C(x)$. The latter rule can be seen as a "macro": it combines into one step the effects of all mentioned inference steps, without expanding the R-successors explicitly.

Computing all relevant "macro" rules is performed by saturating the TBox of KB using *basic superposition* ($\mathcal{BS}$) [2, 16] (a clausal refutation calculus), which

[3] http://jena.sourceforge.net/
[4] http://www.w3.org/TR/owl-semantics/
[5] http://www.w3.org/TR/owl-xmlsyntax/

is implemented in the *Theorem Prover* subcomponent of the Reasoning Engine. Although there are several efficient theorem provers for first-order logic (e.g., Vampire [19], E [20], or Otter [14]), we decided to implement our own theorem prover, due to the following reasons. First, we are unaware of an existing implementation of basic superposition. Second, existing systems usually do not come with a comprehensive API, which makes their integration into other systems difficult. Third, our theorem prover is not used primarily to check TBox inconsistency (an inconsistent TBox is usually a modeling error); rather, it is used to compute all "macro" rules that logically follow from a TBox. Hence, whereas most theorem provers are geared towards unsatisfiable problems, ours is geared towards satisfiable problems. This allows us to make several simplifying assumptions. For example, unlike most existing theorem provers, ours spends very little time in deciding which clause to work on next (in most cases, all clauses must be considered anyway). Fourth, we were not interested in building a general theorem prover; rather, we wanted a prover that implements our algorithm efficiently. This allowed the implementation to be further simplified. In particular, our algorithm must handle only unary and binary literals containing shallow terms, for which unification can be implemented in constant time. Furthermore, clauses can be efficiently indexed using a variant of feature vector indexing [21].

The *Ontology Clausification* subcomponent of the Reasoning Engine is responsible for translating the TBox of a $\mathcal{SHIQ}$ knowledge base KB into a set of first-order clauses. As our experiments confirm, it is very important to reduce the number of clauses produced in the translation. To this purpose, we use several simple optimizations of the clausification algorithm. In particular, if several axioms contain the same nested subconcept, we replace all their occurrences with a new atomic concept. For example, in axioms $C \sqsubseteq \exists R.\exists S.D$ and $E \sqsubseteq \forall T.\exists S.D$ the concept $\exists S.D$ occurs twice, so we replace all its occurrences with a new concept Q. We thus obtain the set of equisatisfiable axioms $C \sqsubseteq \exists R.Q$, $E \sqsubseteq \forall T.Q$, and $Q \sqsubseteq \exists S.D$, which produces fewer clauses than the original one. Another optimization involves functional roles: if R is functional, the existential quantifier in each occurrence of a formula $\exists y : [R(x, y) \wedge C(y)]$ (stemming from a concept $\exists R.C$) can be skolemized using the same function symbol.

The *Disjunctive Datalog Engine* subcomponent of the Reasoning Engine is used for answering queries in the disjunctive datalog program obtained by the reduction. Although several disjunctive datalog engines exist (e.g., DLV [5]), we decided to implement our own engine, due to the following reasons. First, existing engines do not come with a comprehensive API, which makes their integration into other systems difficult. Second, our reduction produces only positive datalog programs—that is, programs without negation-as-failure. We also do not rely on the minimal model semantics of disjunctive datalog. Thus, we can eliminate the minimality test from our implementation and avoid unnecessary overhead. Third, model building is an important aspect of reasoning in disjunctive datalog. To compute the models, disjunctive datalog engines usually ground the disjunctive program. Although this process has been optimized using *intelligent grounding* [6], grounding can be very expensive on large data sets. In contrast,

the models of our programs are of no interest. To avoid grounding, we answer queries using hyperresolution with answer literals. Fourth, disjunctive datalog engines typically do not provide for the first-order equality predicate, which we use to correctly support number restrictions.

Due to space constraints, we cannot present the implementation techniques used in KAON2 in more detail; for more information, please see [15].

4 Benchmarks for ABox and TBox Reasoning

Comparing the performance of DL reasoning systems objectively is difficult because there are no widely accepted benchmarks. Certain ontologies have established themselves as standards for testing TBox reasoning; however, to the best of our knowledge, there are no such standard tests for ABox reasoning. Hence, we constructed our own data set, which we present in this section. The data set is freely available from the KAON2 Web site,[6] and we hope it can be used as a foundation for an extensive DL benchmark suite.

4.1 Selecting Test Data

We wanted to base our tests as much as possible on ontologies created and used in real projects; our intuition was that such ontologies reflect the relevant use cases more accurately than the synthetically generated ones. However, most ontologies currently used in practice seem to fall into two categories: they either have a complex TBox, but no ABox, or they have a large ABox, but a very simple TBox. To obtain tests with interesting TBoxes, we used synthetic ontologies as well. Furthermore, to obtain ABoxes of sufficient size, we applied *replication*—copying an ABox several times with appropriate renaming of individuals in axioms.

One of our goals was to study the impact of various DL constructors on the reasoning performance. In particular, we expected that the presence of equality (stemming from number restrictions), existential quantifiers, and disjunctions will have adverse effects on the performance of reasoning. Therefore, we selected test ontologies that specifically use (a combination of) these constructors.

For some ontologies, the authors also supplied us with the queries used in their projects, which we then reused in our tests. Namely, these queries were usually sent to us because they caused performance problems in practice, so there is reason to believe that they are "hard." Moreover, we expect these queries to better reflect the practical use cases of their respective ontologies.

4.2 Test Ontologies and Queries

VICODI[7] is an ontology about European history, manually created in the EU-funded project VICODI. The TBox is relatively small and simple: it consists of

[6] http://kaon2.semanticweb.org/download/test_ontologies.zip
[7] http://www.vicodi.org/

role and concept inclusion axioms, and of domain and range specifications; furthermore, it does not contain disjunctions, existential quantification, or number restrictions. However, the ABox is relatively large and it contains many interconnected individuals. Because the TBox does not contain existential quantifiers, equality, or disjunctions, it can be converted into a nondisjunctive equality-free datalog program directly, without invoking the reduction algorithm. Hence, query answering for VICODI can be realized using a deductive database only; furthermore, it is possible to deterministically compute the canonical model of the ontology.

With vicodi_0 we denote the ontology from the project, and with vicodi_n the one obtained by replicating n times the ABox of vicodi_0.

From the ontology author we received the following two queries, which are characteristic of the queries used in the project. The first one is a simple concept retrieval, and the second one is a more complex conjunctive query.

$$Q_{V_1}(x) \equiv \mathit{Individual}(x)$$
$$Q_{V_2}(x, y, z) \equiv \mathit{Military\text{-}Person}(x), \mathit{hasRole}(y, x), \mathit{related}(x, z)$$

SEMINTEC is an ontology about financial services, created in the SEMINTEC project[8] at the University of Poznan. Like VICODI, this ontology is relatively simple: it does not use existential quantifiers or disjunctions; it does, however, contain functionality assertions and disjointness constraints. Therefore, it requires equality reasoning, which is known to be hard for deductive databases.

With semintec_0 we denote the ontology from the project, and with semintec_n the one obtained by replicating n times the ABox of semintec_0.

From the ontology author, we obtained the following two queries, which are characteristic of the queries used in the project.

$$Q_{S_1}(x) \equiv \mathit{Person}(x)$$
$$Q_{S_2}(x, y, z) \equiv \mathit{Man}(x), \mathit{isCreditCardOf}(y, x), \mathit{Gold}(y), \mathit{livesIn}(x, z), \mathit{Region}(z)$$

LUBM[9] is a benchmark developed at the Lehigh University for testing performance of ontology management and reasoning systems [7]. The ontology describes organizational structure of universities and it is relatively simple: it does not use disjunctions or number restrictions, but it does use existential quantifiers, so our reduction algorithm must be used to eliminate function symbols. Due to the absence of disjunctions and equality, the reduction algorithm produces an equality-free Horn program. In other words, query answering on LUBM can be performed deterministically.

LUBM comes with a generator, which we used instead of ABox replication to obtain the test data. With lubm_n de denote the ontology obtained from the generator by setting the number of universities to n. The test generator creates many small files; to make these ontologies easier to handle, we merged them into a single file.

[8] http://www.cs.put.poznan.pl/alawrynowicz/semintec.htm
[9] http://swat.cse.lehigh.edu/projects/lubm/index.htm

The LUBM Web site provides 14 queries for use with the ontology, from which we selected the following three. With Q_{L_1} we test the performance of concept retrieval, with Q_{L_2} we test how the performance changes if Q_{L_1} is extended with additional atoms, and with Q_{L_3} we make sure that our results are not skewed by the particular choice of concepts.

$$\begin{aligned} Q_{L_1}(x) &\equiv \mathit{Chair}(x) \\ Q_{L_2}(x,y) &\equiv \mathit{Chair}(x), \mathit{worksFor}(x,y), \mathit{Department}(y), \\ &\quad \mathit{subOrganizationOf}(y, \text{“http://www.University0.edu”}) \\ Q_{L_3}(x,y,z) &\equiv \mathit{Student}(x), \mathit{Faculty}(y), \mathit{Course}(z), \mathit{advisor}(x,y), \\ &\quad \mathit{takesCourse}(x,z), \mathit{teacherOf}(y,z) \end{aligned}$$

Wine[10] is an ontology containing a classification of wines. It uses nominals, which our algorithms cannot handle, so we apply a sound but an incomplete approximation: we replace each enumerated concept $\{i_1, \ldots, i_n\}$ with a new concept O and add assertions $O(i_k)$. This approximation of nominals is incomplete for query answering: for completeness one should further add a clause $\neg O(x) \vee x \approx i_1 \vee \ldots \vee x \approx i_n$; however, doing this would destroy the termination property of our algorithms. The resulting ontology is relatively complex: it contains functional roles, disjunctions, and existential quantifiers.

With wine_0, we denote the original ontology, and with wine_n the one obtained by replicating 2^n times the ABox of wine_0.

Elimination of nominals changes the semantics of most concepts in the knowledge base. Hence, we ran only the following query, which involved computing several nontrivial answers:

$$Q_{W_1}(x) \equiv \mathit{AmericanWine}(x)$$

It is justified to question whether the Wine ontology is suitable for our tests. However, as we already mentioned, we were unable to find an ontology with a complex TBox and an interesting ABox. The approximated Wine ontology was the only one that, at least partially, fulfilled our criteria.

DOLCE[11] is a foundational ontology developed at the Laboratory for Applied Ontology of the Italian National Research Council. It is very complex, and no reasoner currently available can handle it. Therefore, the ontology has been factored into several modules. We used the DOLCE OWL version 397, up to the Common module (this includes the DOLCE-Lite, ExtDnS, Modal and Common modules). Because the ontology does not have an ABox, we used it only for TBox testing.

We have observed that the performance of KAON2 on DOLCE significantly depends on the presence of transitivity axioms. Hence, we included in our benchmarks a version of DOLCE obtained by removing all transitivity axioms.

[10] http://www.schemaweb.info/schema/SchemaDetails.aspx?id=62
[11] http://www.loa-cnr.it/DOLCE.html

Table 2. Statistics of Test Ontologies

KB	$C \sqsubseteq D$	$C \equiv D$	$C \sqcap D \sqsubseteq \bot$	functional	domain	range	$R \sqsubseteq S$	$C(a)$	$R(a,b)$
vicodi_0								16942	36711
vicodi_1								33884	73422
vicodi_2	193	0	0	0	10	10	10	50826	110133
vicodi_3								67768	146844
vicodi_4								84710	183555
semintec_0								17941	47248
semintec_1								35882	94496
semintec_2	55	0	113	16	16	16	6	53823	141744
semintec_3								71764	188992
semintec_4								89705	236240
lubm_1								18128	49336
lubm_2	36	6	0	0	25	18	9	40508	113463
lubm_3								58897	166682
lubm_4								83200	236514
wine_0								247	246
wine_1								741	738
wine_2								1235	1230
wine_3								1729	1722
wine_4								2223	2214
wine_5	126	61	1	6	6	9	9	2717	2706
wine_6								5187	5166
wine_7								10127	10086
wine_8								20007	19926
wine_9								39767	39606
wine_10								79287	78966
dolce	203	27	42	2	253	253	522	0	0
galen	3237	699	0	133	0	0	287	0	0

GALEN[12] is a medical terminology developed in the GALEN project [18]. It has a very large and complex TBox and no ABox, and has traditionally been used as a benchmark for terminological reasoning.

Table 2 shows the number of axioms for each ontology.

5 Performance Evaluation

The main goal of our performance evaluation was to test the scalability of our algorithm—that is, to see how performance of query answering depends on the amount of data and on the complexity of different ontologies. This should give us an idea about the kinds of ontologies that can be efficiently handled using our algorithm. Additionally, we wanted to compare our reasoning algorithm with its tableau counterparts. This goal turned out to be somewhat difficult to achieve. Namely, we are only able to compare *implementations*, and not the algorithms themselves. DL algorithms are complex, and overheads in maintaining data structures or memory management can easily dominate the run time; furthermore, the implementation language itself can introduce limitations that become evident when dealing with large data sets. Therefore, the results we present in this section should be taken qualitatively, rather than quantitatively.

[12] We obtained GALEN through private communication with Ian Horrocks.

5.1 Test Setting

We compared the performance of KAON2 with RACER and Pellet. To the best of our knowledge, these are the only reasoners that provide sound and complete algorithms for $\mathcal{SHIQ}$ with ABoxes.

RACER[13] [8] was developed at the Concordia University and the Hamburg University of Technology, and is written in Common Lisp. We used the version 1.8.2, to which we connected using the JRacer library. RACER provides an optimized reasoning mode (so-called nRQL mode 1), which provides significant performance improvements, but which is complete only for certain types of knowledge bases. When we conducted the evaluation, RACER did not automatically recognize whether the optimized mode is applicable to a particular knowledge base, so we used RACER in the mode which guarantees completeness (so-called nRQL mode 3). Namely, determining whether optimizations are applicable is a form of reasoning which, we believe, should be taken into account in a fair comparison.

Pellet[14] [17] was developed at the University of Maryland, and was the first system to fully support OWL-DL, taking into account all the nuances of the specification. It is implemented in Java, and is freely available with the source code. We used the version 1.3 beta.

We asked the authors of each tool for an appropriate sequence of API calls for running tests. For each reasoning task, we started a fresh instance of the reasoner and loaded the test knowledge base. Then, we measured the time required to execute the task. We made sure that all systems return the same answers.

Many optimizations of tableau algorithms involve caching computation results, so the performance of query answering should increase with each subsequent query. Furthermore, both RACER and Pellet check ABox consistency before answering the first query, which typically takes much longer than computing query results. Hence, starting a new instance of the reasoner for each query might seem unfair. However, we did not yet consider caching for KAON2; furthermore, materialized views were extensively studied in deductive databases, and were successfully applied to ontology reasoning [23]. Also, KAON2 does not perform a separate ABox consistency test because ABox inconsistency is discovered automatically during query evaluation; we consider this to be an advantage of our approach.

Due to these reasons, we decided to measure only the performance of the actual reasoning algorithm, and to leave a study of possible materialization and caching strategies for future work. Since ABox consistency checking is a significant source of overhead for tableau systems, we measured the time required to execute it separately. Hence, in our tables, we distinguish the one-time *setup* time (S) from the query processing time (Q) for Pellet and RACER. This somewhat compensates for the lack of caching: most caches are computed during setup time, so one can expect that subsequent queries will be answered in time similar to the one required for the first query after setup.

[13] http://www.racer-systems.com/

[14] http://www.mindswap.org/2003/pellet/index.shtml

The time for computing the datalog program in KAON2 was comparatively small to the time required to evaluate the program. Therefore, in our test results, we simply included the reduction time into the total query time.

All tests were performed on a laptop computer with a 2 GHz Intel processor, 1 GB of RAM, running Windows XP Service Pack 2. For Java-based tools, we used Sun's Java 1.5.0 Update 5. The virtual memory of the Java virtual machine was limited to 800 MB, and each test was allowed to run for at most 5 minutes.

The results of all tests are shown in Figure 2. Tests which ran either out of memory or out of time are denoted with a value of 10000.

5.2 Querying Large ABoxes

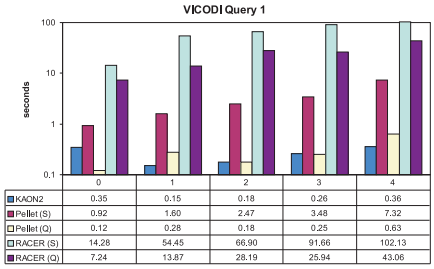
VICODI. The results show that Pellet and RACER spend the bulk of their time in checking ABox consistency by computing a completion of the ABox. Because the ontology is simple, no branch splits are performed, so the process yields a single completion representing a model. Query answering is then very fast in Pellet, as it just requires model lookup. Note that, other than for vicodi_0, the time KAON2 takes to answer queries depends very little on the data size.

It may seem odd that KAON2 takes longer to answer Q_{V_1} on vicodi_0 than on vicodi_1. Repeated tests produced results consistent with the ones reported here. After further analysis, we discovered that this is caused by choosing a suboptimal sideways information passing strategy in the magic sets transformation. We shall try to address this problem in our future research.

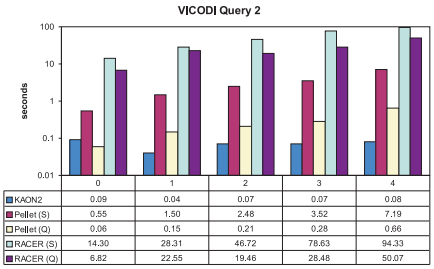
SEMINTEC. The SEMINTEC ontology is roughly of the same size as the VICODI ontology; however, the time that KAON2 takes to answer a query on SEMINTEC is one order of magnitude larger than for the VICODI ontology. This is mainly due to equality, which is difficult for deductive databases. Namely, since any part of the knowledge base can imply two individuals to be equal, techniques such as magic sets that localize reasoning to a portion of the ABox are less effective. Also, notice that all three tools exhibit roughly the same dependency on the size of the data set.

LUBM. As our results show, LUBM does not pose significant problems for KAON2; namely, the translation produces an equality-free Horn program, which KAON2 evaluates in polynomial time. Hence, the time required to answer a query for KAON2 grows moderately with the size of the data set.

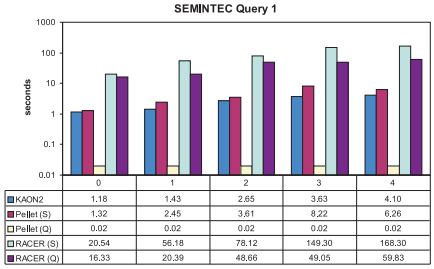
Although LUBM is roughly of the same size as VICODI, both Pellet and RACER performed better on the latter; namely, Pellet was not able to answer any of the LUBM queries within the given resource constraints, and RACER performed significantly better on VICODI than on LUBM. We were surprised by this result: the ontology is still Horn, so an ABox completion can be computed in advance and used as a cache for query answering. By analyzing a run of Pellet on lubm_1 in a debugger, we observed that the system performs disjunctive reasoning (i.e., it performs branch splits). Further investigation showed that this is due to *absorption* [9]—a well-known optimization technique used by all tableau



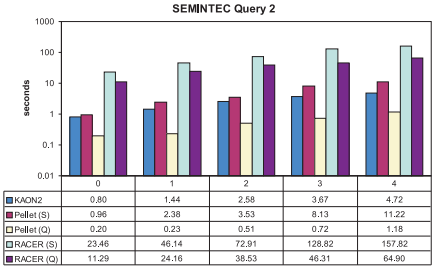
	0	1	2	3	4
KAON2	0.35	0.15	0.18	0.26	0.36
Pellet (S)	0.92	1.60	2.47	3.48	7.32
Pellet (Q)	0.12	0.28	0.18	0.25	0.63
RACER (S)	14.28	54.45	66.90	91.66	102.13
RACER (Q)	7.24	13.87	28.19	25.94	43.06



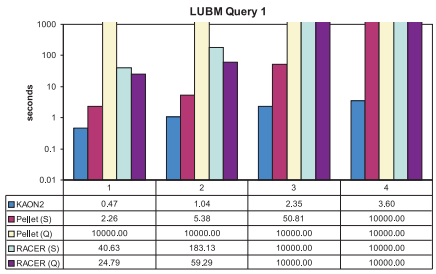
	0	1	2	3	4
KAON2	0.09	0.04	0.07	0.07	0.08
Pellet (S)	0.55	1.50	2.48	3.52	7.19
Pellet (Q)	0.06	0.15	0.21	0.28	0.66
RACER (S)	14.30	28.31	46.72	78.63	94.33
RACER (Q)	6.82	22.55	19.46	28.48	50.07



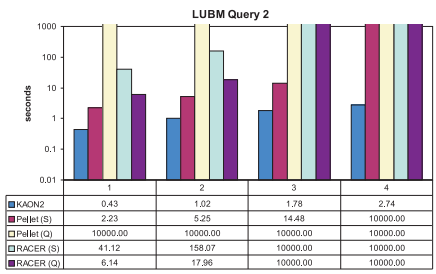
	0	1	2	3	4
KAON2	1.18	1.43	2.65	3.63	4.10
Pellet (S)	1.32	2.45	3.61	8.22	6.26
Pellet (Q)	0.02	0.02	0.02	0.02	0.02
RACER (S)	20.54	56.18	78.12	149.30	168.30
RACER (Q)	16.33	20.39	48.66	49.05	59.83



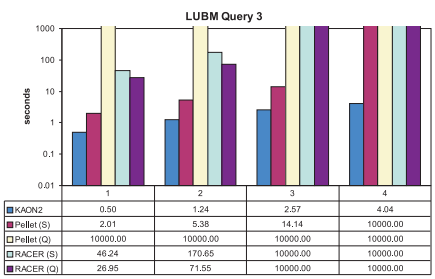
	0	1	2	3	4
KAON2	0.80	1.44	2.58	3.67	4.72
Pellet (S)	0.96	2.38	3.53	8.13	11.22
Pellet (Q)	0.20	0.23	0.51	0.72	1.18
RACER (S)	23.46	46.14	72.91	128.82	157.82
RACER (Q)	11.29	24.16	38.53	46.31	64.90



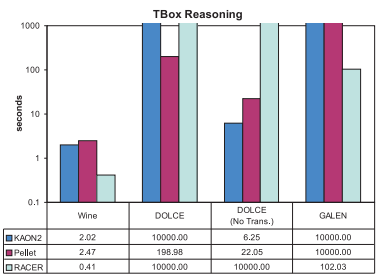
	1	2	3	4
KAON2	0.47	1.04	2.35	3.60
Pellet (S)	2.26	5.38	50.81	10000.00
Pellet (Q)	10000.00	10000.00	10000.00	10000.00
RACER (S)	40.63	183.13	10000.00	10000.00
RACER (Q)	24.79	59.29	10000.00	10000.00



	1	2	3	4
KAON2	0.43	1.02	1.78	2.74
Pellet (S)	2.23	5.25	14.48	10000.00
Pellet (Q)	10000.00	10000.00	10000.00	10000.00
RACER (S)	41.12	158.07	10000.00	10000.00
RACER (Q)	6.14	17.96	10000.00	10000.00



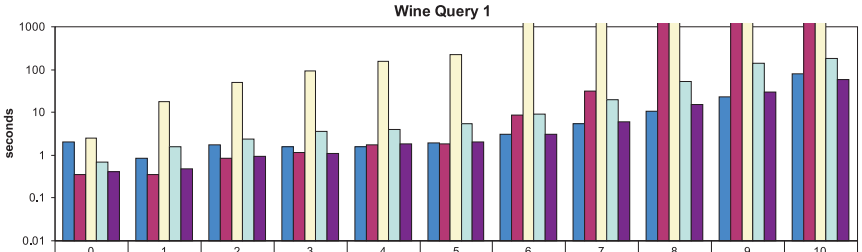
	1	2	3	4
KAON2	0.50	1.24	2.57	4.04
Pellet (S)	2.01	5.38	14.14	10000.00
Pellet (Q)	10000.00	10000.00	10000.00	10000.00
RACER (S)	46.24	170.65	10000.00	10000.00
RACER (Q)	26.95	71.55	10000.00	10000.00



	Wine	DOLCE	DOLCE (No Trans.)	GALEN
KAON2	2.02	10000.00	6.25	10000.00
Pellet	2.47	198.98	22.05	10000.00
RACER	0.41	10000.00	10000.00	102.03

Note: (S) — one-time setup time (including ABox consistency check)
(Q) — time required to process the query

Fig. 2. Test Results



	0	1	2	3	4	5	6	7	8	9	10
KAON2	2.02	0.85	1.72	1.60	1.61	1.95	3.16	5.34	10.78	23.16	80.44
Pellet (S)	0.36	0.36	0.86	1.15	1.76	1.88	8.61	31.79	10000.00	10000.00	10000.00
Pellet (Q)	2.47	17.50	50.44	92.48	154.82	224.77	10000.00	10000.00	10000.00	10000.00	10000.00
RACER (S)	0.68	1.58	2.33	3.53	3.93	5.31	8.91	19.89	51.53	138.30	182.77
RACER (Q)	0.41	0.47	0.94	1.12	1.82	2.07	3.10	6.02	15.35	29.35	59.63

Fig. 2. Test Results (continued)

reasoners. Namely, an axiom of the form $C \sqsubseteq D$, where C is a complex concept, increases the amount of don't-know nondeterminism in a tableau because it yields a disjunction $\neg C \sqcup D$ in the label of each node. If possible, such an axiom is transformed into an equivalent *definition* axiom $A \sqsubseteq C'$ (where A is an atomic concept), which can be handled in a deterministic way. The LUBM ontology contains several axioms that are equivalent to $A \sqsubseteq B \sqcap \exists R.C$ and $B \sqcap \exists R.C \sqsubseteq A$. Now the latter axiom contains a complex concept on the left-hand side of $\sqsubseteq$, so it is absorbed into an equivalent axiom $B \sqsubseteq A \sqcup \forall R.\neg C$. Whereas this is a definition axiom, it contains a disjunction on the right-hand side, and thus causes branch splits. This could perhaps be improved by extending the tableau calculus with an inference rule similar to hyperresolution. Namely, an axiom $B \sqcap \exists R.C \sqsubseteq A$ is equivalent to the clause $B(x) \wedge R(x, y) \wedge C(y) \rightarrow A(x)$. In resolution, one can select the literals on the left-hand side of the implication, which allows the clause to "fire" only if all three literals can be resolved simultaneously. It remains to see whether this is possible in a tableau setting without affecting the correctness and the termination of the calculus.

Wine. The results show that the ontology complexity affects the performance: the ontology wine_0 is significantly smaller than, say, lubm_1, but the times for KAON2 are roughly the same in the two cases. In fact, KAON2 exhibits roughly the same performance as RACER on this test. The degradation of performance in KAON2 is mainly due to disjunctions. On the theoretical side, disjunctions increase the data complexity of our algorithm from P to NP [13]. On the practical side, the technique for answering queries in disjunctive programs used in KAON2 should be further optimized.

5.3 TBox Reasoning

Our TBox reasoning tests clearly show that the performance of KAON2 lags behind the performance of the tableau reasoners. This should not come as a surprise: in the past decade, many optimization techniques for TBox reasoning

were developed for tableau algorithms, and these techniques are not directly applicable to the resolution setting. Still, KAON2 can classify DOLCE without transitivity axioms, which is known to be a fairly complex ontology. Hence, we believe that developing additional optimization techniques for resolution algorithms might yield some interesting and practically useful results.

We analyzed the problems which KAON2 failed to solve. Roughly speaking, all these problems contained many concepts of the form $\exists R.C$ and $\forall R.D$ involving the same role R. The first type of concepts produces clauses with a literal $R(x, f(x))$, whereas the second type of clauses produces clauses with a literal $\neg R(x, y)$. Obviously, these clauses can participate in a quadratic number of resolution inferences in the beginning of a saturation, which eventually leads to an exponential blowup. This explains why KAON2 is not able to classify the original DOLCE ontology, but why it works well if the transitivity axioms are removed: the approach for dealing with transitivity in KAON2 introduces axioms that, when clausified, produce many clauses with such literals.

6 Conclusion

In this paper, we described KAON2—a DL reasoner based on a novel reasoning algorithm that allows for the application of optimization techniques from deductive databases to DL reasoning. To verify our conjecture that such algorithms will scale well in practice, we created a set of benchmarks and conducted a thorough performance evaluation. The results were roughly in line with our expectations: for ontologies with rather simple TBoxes, but large ABoxes, our algorithm indeed provides good performance; however, for ontologies with large and complex TBoxes, existing algorithms still provide superior performance.

For our future work, the main challenge is to extend the reduction algorithm to handle nominals. Furthermore, we believe that optimizations based on ABox statistics will provide further significant improvements. Finally, we shall investigate further optimizations of TBox reasoning.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, January 2003.
2. L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic Paramodulation. *Information and Computation*, 121(2):172–192, 1995.
3. S. Bechhofer, R. Volz, and P. W. Lord. Cooking the Semantic Web with the OWL API. In *Proc. ISWC 2003*, volume 2870 of *LNCS*, pages 659–675, Sanibel Island, FL, USA, October 20–23 2003. Springer.
4. C. Cumbo, W. Faber, G. Greco, and N. Leone. Enhancing the Magic-Set Method for Disjunctive Datalog Programs. In *Proc. ICLP 2004*, volume 3132 of *LNCS*, pages 371–385, Saint-Malo, France, September 6–10 2004. Springer.
5. T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative problem-solving using the DLV system. *Logic-Based Artificial Intelligence*, pages 79–103, 2000.

6. T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A Deductive System for Non-Monotonic Reasoning. In *Proc. LPNMR '97*, volume 1265 of *LNAI*, pages 364–375, Dagstuhl, Germany, July 28–31 1997. Springer.
7. Y. Guo, Z. Pan, and J. Heflin. An Evaluation of Knowledge Base Systems for Large OWL Datasets. In *Proc. ISWC 2004*, volume 3298 of *LNCS*, pages 274–288, Hiroshima, Japan, November 7–11 2004. Springer.
8. V. Haarslev and R. Möller. RACER System Description. In *Proc. IJCAR 2001*, volume 2083 of *LNAI*, pages 701–706, Siena, Italy, June 18–23 2001. Springer.
9. I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, UK, 1997.
10. I. Horrocks and P. F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. *Journal of Web Semantics*, 1(4):345–357, 2004.
11. I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Very Expressive Description Logics. *Logic Journal of the IGPL*, 8(3):239–263, 2000.
12. U. Hustadt, B. Motik, and U. Sattler. Reducing $\mathcal{SHIQ}^-$ Description Logic to Disjunctive Datalog Programs. In *Proc. KR 2004*, pages 152–162, Whistler, Canada, June 2–5, 2004 2004. AAAI Press.
13. U. Hustadt, B. Motik, and U. Sattler. Data Complexity of Reasoning in Very Expressive Description Logics. In *Proc. IJCAI 2005*, pages 466–471, Edinburgh, UK, July 30–August 5 2005. Morgan Kaufmann Publishers.
14. W. W. McCune. OTTER 3.0 Reference Manual and Guide. Technical Report ANL-94/6, Argonne National Laboratory, January 1994.
15. B. Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Univesität Karlsruhe, Germany, 2006.
16. R. Nieuwenhuis and A. Rubio. Theorem Proving with Ordering and Equality Constrained Clauses. *Journal of Symbolic Computation*, 19(4):312–351, 1995.
17. B. Parsia and E. Sirin. Pellet: An OWL-DL Reasoner. Poster, In Proc. ISWC 2004, Hiroshima, Japan, November 7–11, 2004.
18. A. L. Rector, W. A. Nowlan, and A. Glowinski. Goals for concept representation in the galen project. In *Proc. SCAMC '93*, pages 414–418, Washington DC, USA, November 1–3 1993. McGraw-Hill.
19. A. Riazanov and A. Voronkov. The design and implementation of VAMPIRE. *AI Communications*, 15(2–3):91–110, 2002.
20. S. Schulz. E—A Brainiac Theorem Prover. *AI Communications*, 15(2–3):111–126, 2002.
21. S. Schulz. Simple and Efficient Clause Subsumption with Feature Vector Indexing. In *Proc. ESFOR, IJCAR 2004 Workshop*, Cork, Ireland, July 4–8 2004.
22. D. Tsarkov and I. Horrocks. Ordering Heuristics for Description Logic Reasoning. In *Proc. IJCAI 2005*, pages 609–614, Edinburgh, UK, July 30 – August 5 2005. Morgan Kaufmann Publishers.
23. R. Volz. *Web Ontology Reasoning With Logic Databases*. PhD thesis, Universität Fridericiana zu Karlsruhe (TH), Germany, 2004.