Optimizing Query Answering in Description Logics using Disjunctive Deductive Databases

Boris Motik Raphael Volz
Alexander Maedche
FZI Research Center for Information Technologies
at the University of Karlsruhe,
D-76131 Karlsruhe, Germany
{motik,volz,maedche}@fzi.de

Abstract

Motivated by the possibilities of applying deductive database technology for efficient query answering in description logics, we present a translation operator μ that transforms non-recursive $\mathcal{ALC}$ ontologies into a disjunctive deductive database. Contrary to our previous work, in this paper we focus on handling negation, disjunction and existential quantifiers, which cannot be handled by deductive databases in a straightforward manner. We present a performance evaluation of our approach, confirming the intuition that techniques for optimizing query answering in disjunctive deductive databases may improve query answering in description logics.

1 Introduction

The inference mechanisms employed in the state-of-the-art description logic systems are typically based on refutation theorem proving techniques using tableau calculus. This calculus provides a decision procedure for one concept-instance pair. However, it is generally acknowledged that efficient query answering over ontologies with large number of instances requires management of information in sets. Hence, in spite of various optimization techniques (e.g. [15]), performance of query answering in description logic systems is suboptimal for ontologies containing large number of instances.

Disjunctive deductive databases [7] have been investigated as an alternative knowledge representation formalism. Recently, several advanced techniques for query answering in disjunctive databases have been presented. This includes the ordering refinement of hyperresolution [4] and disjunctive magic sets transformation [11]. It has been shown experimentally that these techniques significantly improve query answering in large databases.

Paper Contribution. Motivated by the prospect of improving query answering in description logics, in this paper we extend our previous work from [12] and provide means for handling central description logic constructs using disjunctive deductive databases, namely

classical negation and disjunction. Since $\mathcal{ALC}$ is the foundation of almost all other more expressive logics, we are able to bridge the most critical differences between them and deductive databases. Additionally, our work provides the following benefits:

- We can improve the performance of DL reasoners by employing evaluation techniques developed in the field of deductive databases.
- If an ontology uses constructs within the Horn fragment, our approach does not introduce any performance penalty. In this way, performance penalty is paid only for features actually used.
- Our approach presents a framework for interoperability between description logics and deductive databases.

We implemented our approach in KAON – the ontology management infrastructure developed by FZI and AIFB at the University of Karlsruhe. We compared the performance of our system with RACER [13] – a state-of-the-art description logics reasoning system. In many cases our approach outperforms RACER in query answering on several orders of magnitude. While we are uncertain what the exact causes for such behavior are, we attempted to provide a plausible explanation.

We assume that the reader is familiar with the fundamentals of description logics (for an introduction see e.g. [1]) and disjunctive deductive databases (for an introduction see e.g. [7]). The rest of this paper is structured as follows. In Section 2 we present our approach for handling description logics using disjunctive deductive databases. In Section 3 we present the results of the performance evaluation of our approach. Finally, before we conclude, in Section 4 we present an overview of the related work.

2 A-Box Query Answering using Deductive Databases

In this section we examine how query evaluation techniques of deductive databases can be applied to query answering in $\mathcal{ALC}$ description logic A-Boxes. An excellent introduction to disjunctive databases is given in [7], but without allowing for rules with function symbols. However, it is straightforward to extend the minimal or stable model semantics to models with functional terms. The problems related to query termination in such models are partially addressed in this section and largely left for future work.

2.1 Translating $\mathcal{ALC}$ Ontology into Deductive Database

Extending the work from [12], in this subsection we present the operator μ that can be applied to any $\mathcal{ALC}$ ontology O to produce a disjunctive deductive database $\mu[O]$.

In our presentation, $[C]$ we denote the predicate with the name equal to the concept expression C. The predicate HU contains the Herbrand's universe of the deductive database.

Normalizing Concept Expressions. Each concept expression in O is first translated into negation-normal form, where negation symbols occur directly in front of the atomic concept names. This can be done by pushing negation inwards through application of identities $\neg(C \sqcap D) \equiv \neg C \sqcup \neg D$, $\neg(C \sqcup D) \equiv \neg C \sqcap \neg D$, $\neg\neg C \equiv C$, $\neg\exists R.C \equiv \forall R.\neg C$ and $\neg\forall R.C \equiv$

$\exists R.\neg C$. This transformation can always be performed, but may result in a concept expression with size exponential with respect to the size of the original expression.

Translating A-Box Assertions. Each A-Box assertion is simply translated into equivalent fact in the deductive database, according to the first section of Table 1.

Translating T-Box Axioms. For each T-Box inclusion axiom, one or two rules may be generated, according to the second section of Table 1.

Expanding Concept Expressions. Rules and facts generated in previous steps may contain complex predicates equal to concept concept expressions (expressions which are not atomic concept names or negations of atomic concept names). These must be further expanded. The underline in the generated rules or facts means that concept expression hasn't been expanded yet. The expansion is performed iteratively, by choosing one underlined concept expression. If the expression is complex (i.e. non-atomic concept name), then the corresponding action from the Table 1 is performed. Finally, the underline from the chosen concept expression is removed. The process is performed until there are no underlined concept expressions.

The third section of Table 1 specifies how rules containing certain concept expressions are changed. The fourth section of the table specifies how additional rules are generated for concept expressions occurring in the rule head. Finally, the fifth section of the table specifies how rules are generated for concept expressions occurring in the rule body. In the table H denotes a disjunction of head literals, and B denotes the conjunction of body literals. The function symbol f should in each expansion be replaced by a new function symbol. Symbol $\neg$ should be interpreted as classical negation and is dealt with later.

It is obvious that this algorithm terminates – in each step the complexity of the expanded concept expression is decreased. Therefore, after a finite number of steps all complex concept expressions will be expanded to the atomic ones. Further, each concept subexpression of O is visited in the translation process at most once, so the translation algorithm is linear in the number of concept subexpressions in O.

Adding Classical Negation. The program obtained by expanding concept expressions according to above rules may contain classically negated atoms. Such programs can be handled in the style of Gelfond and Lifschitz [10]. We extend their approach by further axiomatizing the usual first-order semantics.

Each literal of the form $\neg P(\mathbf{X})$ should be replaced with literal $[\neg P](\mathbf{X})$, where $[\neg P]$ is a new predicate symbol. Intuitively, $[\neg P]$ will contain those facts for which it can be proven that P doesn't hold. We say that P and $[\neg P]$ are complementary pairs of predicates. For each complementary pair of predicates where both predicates occur in the program, following rules should be added:

$$P(X) \vee [\neg P](X) \text{ :- } HU(X)$$
$$\text{:- } P(X), [\neg P](X)$$

The first rule axiomatizes the fact that for each individual of the domain, either P or $[\neg P]$ should hold, whereas the second rule says that for no individual both P and $[\neg P]$ can hold.

	For A-Box Axiom...	...generate fact:
1.	$\mu[C(a)]$	$\underline{[C]}(a)$
2.	$\mu[R(a,b)]$	$R(a,b)$
	For T-Box Axiom...	...generate rule:
3.	$C \sqsubseteq D$	$\underline{[D]}(X)$:- $\underline{[C]}(X)$
4.	$C \equiv D$	$\underline{[D]}(X)$:- $\underline{[C]}(X)$ $\underline{[C]}(X)$:- $\underline{[D]}(X)$
	For rule of the form...	...do the following:
5.	$\underline{[\top]}(X) \vee H$:- B	Delete the rule.
6.	H :- $\underline{[\top]}(X), B$	If variable X occurs in H but not in B, replace the rule with H :- $HU(X), B$ otherwise replace it with H :- B
7.	$\underline{[\bot]}(X) \vee H$:- B	Replace the rule with H :- B
8.	H :- $\underline{[\bot]}(X), B$	Delete the rule.
9.	$\underline{[\neg A]} \vee H$:- B	Replace the rule with $\neg\underline{[A]} \vee H$:- B
10.	H :- $\underline{[\neg A]}, B$	Replace the rule with H :- $\neg\underline{[A]}, B$
	For expression in the head...	...add rule:
11.	$\underline{[C_1 \sqcap \ldots \sqcap C_n]}$	$\underline{[C_1]}(X)$:- $[C_1 \sqcap C_2 \sqcap \ldots \sqcap C_n](X)$ $\vdots$ $\underline{[C_n]}(X)$:- $[C_1 \sqcap C_2 \sqcap \ldots \sqcap C_n](X)$
12.	$\underline{[C_1 \sqcup \ldots \sqcup C_n]}$	$\underline{[C_1]}(X) \vee \ldots \vee \underline{[C_n]}(X)$:- $[C_1 \sqcup \ldots \sqcup C_n](X)$
13.	$\underline{[\exists R.C]}$	$R(X, f(X))$:- $[\exists R.C](X)$ $\underline{[C]}(f(X))$:- $[\exists R.C](X)$ $\overline{HU}(f(X))$:- $[\exists R.C](X)$
14.	$\underline{[\forall R.C]}$	$\underline{[C]}(Y)$:- $R(X,Y), [\forall R.C](X)$
	For expression in the body...	...add rule:
15.	$\underline{[C_1 \sqcap \ldots \sqcup C_n]}$	$[C_1 \sqcap \ldots \sqcup C_n](X)$:- $\underline{[C_1]}(X), \ldots, \underline{[C_n]}(X)$
16.	$\underline{[C_1 \sqcup \ldots \sqcup C_n]}$	$[C_1 \sqcup \ldots \sqcup C_n](X)$:- $\underline{[C_1]}(X)$ $\vdots$ $[C_1 \sqcup \ldots \sqcup C_n](X)$:- $\underline{[C_n]}(X)$
17.	$\underline{[\exists R.C]}$	$[\exists R.C](X)$:- $R(X,Y), \underline{[C]}(Y)$
18.	$\underline{[\forall R.C]}$	$R(X, f(X))$:- $\neg[\forall R.C](X)$ $\neg\underline{[C]}(f(X))$:- $\neg[\forall R.C](X)$ $\overline{HU}(f(X))$:- $\neg[\forall R.C](X)$

Table 1: Translation Rules

2.2 Example

To aid the understanding of the approach, we present a simple example. Consider an $\mathcal{ALC}$ ontology from Table 2. The left-hand side of the table is the TBox, whereas the right-hand side of the table is the ABox.

(T1)	Person $\equiv$ Man $\sqcup$ Woman	(A1)	$\neg$Man(Jane)
(T2)	Woman $\sqcap$ Man $\sqsubseteq \bot$	(A2)	Woman(Jill)
(T3)	Mother $\equiv$ Woman $\sqcap \exists$hasChild.Person	(A3)	hasChild(Jill, Jane)
(T4)	Father $\equiv$ Man $\sqcap \exists$hasChild.Person		

Table 2: Example $\mathcal{ALC}$ Ontology

Table 3 shows rules generated by translating the $\sqsupseteq$ side of axiom (T3). Rule (R1) is generated by applying translation 3 from Table 1. Predicates on either side of the implication sign are underlined, which means that they need to be further expanded. Since predicate [Mother] is an atomic one, it can't be expanded, so underline can simply be removed. However, the predicate [Woman $\sqcap \exists$hasChild.Person] is not atomic and occurs in the head, so its underline can be removed and transformation 11 applied, generating rules (R2) and (R3). Now predicate [Woman] is atomic, so the underline can simply be removed. However, the predicate [$\exists$hasChild.Person] is not atomic and occurs in the head, so its underline can be removed and transformation 13 applied, generating rules (R4), (R5) and (R6). Finally, since predicate [Person] is atomic, its underline can simply be removed. The expansion of the axiom (T3) is thus complete. Other rules from the example ontology can be generated in a similar way. Finally, concept Man appears negated in the ABox axiom (A1). Therefore, a predicate [$\neg$Man] is introduced and axiomatized through rules (R7) and (R8).

(R1)	[Woman $\sqcap \exists$hasChild.Person](X) :- [Mother](X)
(R2)	[Woman](X) :- [Woman $\sqcap \exists$hasChild.Person](X)
(R3)	[$\exists$hasChild.Person](X) :- [Woman $\sqcap \exists$hasChild.Person](X)
(R4)	hasChild$(X, f(X))$:- [$\exists$hasChild.Person](X)
(R5)	[Person]$(f(X))$:- [$\exists$hasChild.Person](X)
(R6)	$HU(f(X))$:- [$\exists$hasChild.Person](X)
(R7)	$[Man](X) \vee [\neg Man](X)$:- $HU(X)$
(R8)	:- $[Man](X), [\neg Man](X)$

Table 3: Translation into Disjunctive Deductive Database

2.3 Evaluating Transformed Programs

In this section we explain how transformed programs may be efficiently evaluated. We are primarily interested in query answering, that is, computing the extension of some concept or role predicate Q. For this purpose, we use the ordered hyperresolution technique from [4] for that purpose. In this setting one may define an arbitrary order $\prec$ of ground literals of

the program, with the only constraint that all literals containing predicate Q follow all other literals.

Definition 1 (Disjunctive Fact) *A disjunctive fact is a set of ground literals of the form $\{L_1, \ldots, L_n\}$, which is often also written as $L_1 \vee \ldots \vee L_n$. Literal L_i of a disjunctive fact $F = \{L_1, \ldots, L_n\}$ is the active literal (written act$(F) = L_i$) if for all $j \neq i$ $L_i \prec L_j$. For some set of disjunctive facts $\mathcal{F}$, by act$(\mathcal{F}) = \{act(F) \mid F \in \mathcal{F}\}$ we denote the set of all active literals of $\mathcal{F}$.*

The ordering makes sure that each disjunctive fact has exactly one active literal. Now we define the set of immediate consequences of some program on the set of disjunctive facts F. Intuitively, immediate consequences contain all facts that can be derived from F through hyperresolution on the active literal:

Definition 2 (Immediate Consequence) *The immediate consequence of P on F is the following set of disjunctive facts is the following set:*

$T_P(\mathcal{F}) = \mathcal{F} \cup \{F \mid$

- *there is a rule instance $A_1 \vee \ldots \vee A_n$:- $B_1, \ldots, B_m$ in $ground(P)$ with $n \geq 0$,*
- *there are disjunctive facts $F_1, \ldots, F_m \in \mathcal{F}$, $B_i = act(F_i)$,*
- $F = \{A_1, \ldots, A_n\} \cup \bigcup_{i=1}^{m}(F_i - \{B_i\})$

$\}$

We denote with $T_P^\infty(\mathcal{F})$ the least fixpoint of T_P, that is, the set obtained by successive application of T_P until noting changes. In [4] it has been shown that $T_P^\infty(\mathcal{F})$ is complete in the sense that it will contain all disjunctions of the form $Q_1 \vee \ldots \vee Q_n$. The answer to the query can then be obtained by selecting all singleton disjunctive facts containing Q alone.

As discussed in [4], the presented evaluation strategy results in reasonable performance it the number of disjunctions is not too big. This is mainly due to the application of the ordering refinement, which reduces the number of irrelevant disjunctions derived. Further, the presented evaluation strategy can be used in combination with disjunctive magic sets rewriting, thus resulting in further performance increase.

2.4 Limitations of Evaluation Approach

Transformed program $\mu[O]$ may contain function symbols. In that case, the Herbrand's universe will be infinite. Thus, the minimal model of the program may also become infinite, making it impossible to compute it. Consider the ontology containing the only T-Box axiom $A \sqsubseteq \exists R.A$. This axiom is translated into these rules:

$$[\exists R.A](X) \text{ :- } [A](X)$$
$$R(X, f(X)) \text{ :- } [\exists R.A](X)$$
$$[A](f(X)) \text{ :- } [\exists R.A](X)$$

Now if the A-Box contains the assertion $[A](a)$, the minimal model contains facts of the form $[A](f(a))$, $[A](f(f(a)))$ etc. Note, however, that the minimal model contains a finite subset of all simple facts we may be interested in, but additionally it contains an infinite number of facts about functional terms.

Currently, the results from this paper can be used only if function symbols don't occur in rules occurring in a cycle of a dependency graph. In the rest of this section we describe briefly our approach for overcoming this difficulty.

Many modal logics can be decided by a saturation-based resolution procedure. For example, in [9] such a procedure has been has been presented for the modal logic K4, which includes transitive frames. The main feature of this procedure is that the saturated sets of clauses for any logical formula are finite and don't include functional terms of depth more than one. We are currently building a similar resolution procedure for $\mathcal{SHIQ}$ description logics. With such a procedure in place, we conjecture it will be possible to translate easily saturated clause set into a disjunctive program without function symbols which can then be evaluated using presented techniques.

3 Performance Evaluation

In order to show the benefits of our approach, we prototypically implemented our approach in KAON – KArlsruhe ONtology management suite. We conducted a series of performance tests, the results of which we show in this section. We executed the tests with Racer version 1.7 and our prototype. We didn't use the FaCT system [14] since it doesn't support A-Box assertions.

3.1 Test Setting

About Tools. Racer [13] was chosen as the state-of-the-art description logic reasoning system employing the tableau decision procedure as the inference mechanism. The implementation language of Racer is LISP. The implementation language of KAON and of our prototype is Java.

Test Assumptions. Many description logic systems use caching extensively in order to speed up query processing. For example, once Racer computes the extension of some concept, it caches the results, so the next time the same query is issued, it is answered almost immediately. We decided not to take this into account since we wanted to measure the performance of query answering alone. It is quite obvious that, if the answer to the query is cached, query answering will be fast. Moreover, Racer doesn't perform incremental maintenance of query answers. We have observed that whenever the A-Box is changed, even if the change doesn't affect the result of the query, Racer forgets all cached information and answering the query takes the same amount of time as the first time.

Test Procedure. Each test is characterized by a certain ontology structure and a concept whose extension is to be read. The ontology structure has been generated for different input parameters, resulting in ontologies of different sizes. Obtained ontologies have then been loaded in each of the tools and the query has been executed. The average of five such invocations has been taken as the performance measure for each test. If executing the query took

more than 15 minutes, the test was interrupted – results of such tests are denoted with MAX in the table.

Test Platform. We performed the tests on a standard PC with Pentium III processor running at 1.1 GHz, 380 MB of RAM running Windows XP operating system. Tests were written in Java and run using Sun's JDK version 1.4.1_01. Communication with Racer was done using JRacer library.

Finally, before presenting the test results, we want to stress that we measured the performance of concrete tools. Although the algorithms used by all mentioned systems are certainly important, the overall performance of the system is influenced by many other factors as well, such the quality of the implementation or the language used to implement the system. It is virtually impossible to exclude these factors from the performance measurement.

3.2 Measurement Results

First we give an overview of the types of tests we conducted. The results of tests 1 to 6 are presented in Figure 1, whereas the results of test 7 are presented in Figure 2.

In describing tests we use *D* to denote the depth of the concept tree, *NS* to denote the number of subconcepts at each level in the tree, *NI* to denote the number of instances per concept and *P* to denote the number of properties.

Test 1. The goal of this test was to see how the very basic tasks of traversing the concept hierarchy are handled in ontologies with larger number of properties. The ontology structure was a symmetric tree of directly classified concepts with one property per concept, which was instantiated for every third instance of the concept pointing to the next instance. However, the properties were not mentioned in concept definitions (i.e. they were not relevant to the query at all). This test didn't include disjunction or negation. The query involved computing the extension of one of the first-level concepts. The test was performed for *D* = 3, 4, 5; *NS* = 5; *NI* = 10; *P* = 155, 780, 3905. Query answering was performed using magic sets transformation.

Test 2. In the previous test we observed that the performance of Racer depended on the number of property instances, even if these are not mentioned in concept definitions. Hence, in this test we wanted to see whether smaller number of properties, but larger number of property instances will make a difference. The ontology structure from Test 1 was extended with a fixed number of properties. Each instance was connected with the previously generated instance through one property. The test was performed for *D* = 3, 4, 5; *NS* = 5; *NI* = 10; *P* = 211. Query answering was performed using magic sets transformation.

Test 3. The goal of this test was to test answering a simple conjunctive query. The ontology structure was identical to the one from Test 3, but the query was $\mathrm{c}_1 \sqcap \exists \mathrm{p}_0.\mathrm{c}_{12}$. The test for performed for *D* = 3, 4; *NS* = 5; *NI* = 10; *P* = 3. Query answering was performed using magic sets transformation.

Test 4. The goal of this test was to see how large number of disjunctive axioms are handled. The ontology structure was an inverse tree of concepts where j-th concept at level i was defined using the following axiom: $\mathrm{c}_{i,j} \sqsubseteq \mathrm{c}_{i-1,j} \sqcup \mathrm{c}_{i-1,j+1}$. Hence, the ontology had lots of disjunctions which weren't relevant for the query. The query involved computing the extension of concept $\mathrm{c}_{3,1}$. Query answering was performed without the magic sets transformation, since the intention was to compare only the performance of handling disjunctions.

Test 5. The goal of this test was to compare both tools on a concrete ontology. We used a simple ontology describing relationships between documents, processes and persons that we built in one of our industry projects. The ontology was small: it contained 24 concepts, 29 properties and 513 instances. To the best of our efforts we were unable to obtain a larger ontology which is generated by people (rather than converted from some existing data source). Also, in this test we didn't include any disjunction or negation. The query involved computing an extension of one concept and resulted in 12 instances. Query answering was performed using the magic sets transformation, since we wanted to show how ontologies in the Horn subset can be handled efficiently using existing techniques.

Test 6. The goal of this test was to compare both tools on a concrete ontology but containing disjunctive information. We extended the ontology from test 5 with a couple of integrity constraints, axioms having disjunction in the head and axioms involving negation. Also, we added automatically generated instances to the ontology, so it contained 25 concepts, 29 properties and 1314 instances. The query involved computing an extension of one concept and resulted in 484 instances. Query answering wasn't performed using the magic sets transformation – we analyzed the efficiency of magic sets in a separate test.

Test 7. In this test we analyzed the potential in applying the (disjunctive) magic sets transformation to improve query answering. Hence, we repeated test 4, but computed extension of concepts at different levels in the tree. By this we wanted to show that magic sets are most effective for identifying the part of the ontology relevant for the query and evaluating the query only in this segment of the ontology. By increasing the depth, the relevant part of the ontology increased, so magic sets transformation has less effect. Finally, at some point magic sets actually introduced an overhead, rather than improving the performance.

3.3 Discussion

From the results one can observe that the performance of Racer in all tests is significantly worse than the performance of KAON. We anticipate that this is mainly due to the fact that tableau reasoning procedure provides a proof or a refutation for one concept-instance pair, whereas bottom-up computations works with sets of instances.

With respect to the disjunctive queries, we anticipate that ordering refinement of hyperresolution has significant influence on improving the performance. This optimization basically allows us not to have to compute entire minimal models, but to work only with parts of models leading to the answer of the query.

Finally, one may see that magic sets are a promising technique in optimizing query answering. This is especially true if queries are localized to a subset of the ontology – magic sets can efficiently select the relevant part of the ontology. Another important aspect of magic sets optimization, namely sideways information passing, is less important in management of description logics: most predicates are unary, so passing of bindings occurs less often.

4 Related Work

Our work conceptually follows from the relationship between description logic and FOL [2]. An axiomatization of DAML+OIL, the precursor of OWL, was given in [8]. Their axiomati-

T	Set	C	I	P	Racer	KAON
1	1	155	1550	155	6.99	**0.21**
	2	780	7800	780	98.56	**0.72**
	3	3905	39050	3905	MAX	**4.24**
2	1	155	1550	211	5.39	**0.39**
	2	780	7800	211	MAX	**0.88**
	3	3905	39050	211	MAX	**4.61**
3	1	155	1550	3	34.30	**0.82**
	2	780	7800	3	MAX	**3.07**
4	1	46	2300	0	10.39	**0.71**
	2	106	5300	0	64.45	**2.49**
	3	191	9550	0	168.82	**8.82**
5		24	513	29	4.24	**0.32**
6		25	1314	29	7.16	**0.71**

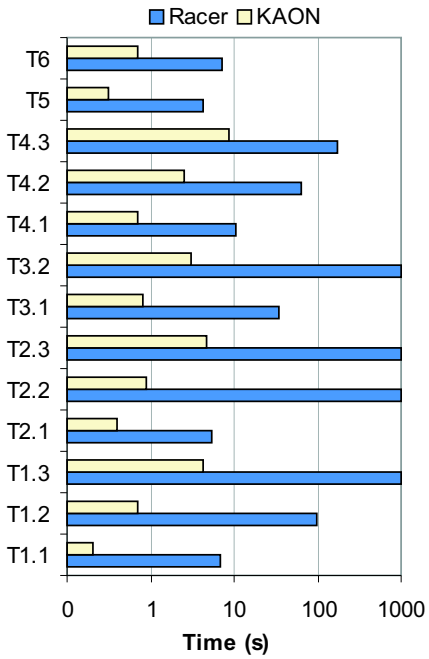

Figure 1: Results of Tests 1 to 6 (in seconds)

Level	With Magic (s)	No Magic (s)
1	1.12	8.48
2	1.47	8.52
3	3.28	8.58
4	7.29	8.64
5	15.04	8.88

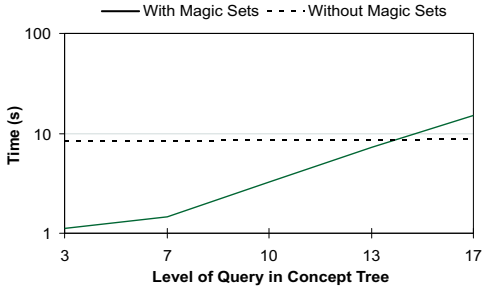

Figure 2: Results of Test 7 (in seconds)

zation language is the Knowledge Interchange Format (KIF), so the axiomatization is not directly executable using logic programming systems, but a theorem prover is needed. Because of that, application of optimization techniques, such as magic sets, is not straight-forward.

Several systems, such as CARIN [16] or AL-log [6], attempted the integration of description logic with datalog rules. These systems don't provide a translation of one formalism into another. Rather, they investigate which primitives from both formalisms can be successfully combined, resulting in a decidable system. Both systems rely use tableau.

There were several attempts at providing methods for integrating description logics reasoners with other systems storing information about instances. In [3] an approach for translating description logic concepts descriptions into SQL queries has been presented. A similar, more general framework for translating description logics into queries over different information servers was presented in [5]. Both approaches are based on the CLASSIC [17] description logics, which is weaker in expressivity than $\mathcal{ALC}$. In particular, it doesn't allow usage of disjunction and negation. Further, both approaches mention the problem that some rules might be translated to unsafe queries, but present no solution.

5 Conclusion

Motivated by the prospects of answering queries over description logics using disjunctive deductive databases, we have presented the operator μ that translates any $\mathcal{ALC}$ knowledge base to a disjunctive deductive database. Our approach is currently applicable to cases where existential quantifiers symbols don't occur in heads of axioms of cyclical definitions. We implemented a prototype system and have experimentally compared performance of our approach with that of traditional description logic systems. We have found out that for usual description logics databases our approach behaves on the orders of magnitude better.

In future we primarily plan to address the problem of function symbols in recursive rules. Further, we'd like to extend our translation with equality, which will enable us to capture cardinality constraints – another very expressive feature of many description logics. Also, we are interested in investigating how non-monotonic features, such as default inheritance, can be added to description logics in this framework.

References

[1] A. Borgida. Description logics are not just for the FLIGHTLESS-BIRDS: A new look at the utility and foundations of description logics. Technical Report DCS-TR-295, Department of Computer Science, Rutgers University, 1992.

[2] A. Borgida. On the Relative Expressiveness of Description Logics and Predicate Logics. *Artificial Intelligence*, 82(1-2):353–367, 1996.

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

[4] S. Brass and U. W. Lipeck. Generalized Bottom-Up Query Evaluation. In A. Pirotte, C. Delobel, and G. Gottlob, editors, *Advances in Database Technology - Proceedings EDBT'92*, pages 88–103, Berlin, 1992. Springer-Verlag.

[5] P. T. Devanbu. Translating Description Logics to Information Server Queries. In *CIKM 93, Proc. of the 2nd Int'l Conf. on Information and Knowledge Management*, pages 256–263, Washington, DC, USA, November 1993. ACM.

[6] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Al-log: integrating datalog and description logics. *Journal of Intelligent Information Systems*, 10:227–252, 1998.

[7] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 1997.

[8] R. Fikes and D. McGuiness. An axiomatic semantics for RDF, RDF Schema and DAML+OIL. Technical Report KSL-01-01, KSL, Stanford University, 2001.

[9] H. Ganzinger, U. Hustadt, C. Meyer, and R. A. Schmidt. A Resolution-Based Decision Procedure for Extensions of K4. In M. Zakharyaschev, K. Segerberg, M. de Rijke, and H. Wansing, editors, *Advances in Modal Logic, Volume 2*, volume 119 of *Lecture Notes*, pages 225–246. CSLI Publications, Stanford, 2001.

[10] Michael Gelfond and Vladimir Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9(3/4):365–386, 1991.

[11] S. Greco. Binding Propagation Techniques for the Optimization of Bound Disjunctive Queries. *IEEE Transactions on Knowledge and Data Engineering*, 15(2):717–736, March/April.

[12] B. Grossof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *Proceedings of WWW 2003*, Budapest, Hungary, May 2003.

[13] V. Haarslev and R. Möller. RACER System Description. *Lecture Notes in Computer Science*, 2083, 2001.

[14] I. Horrocks. The FaCT System. In *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98*, pages 307–312. Springer-Verlag.

[15] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

[16] A. Y. Levy and M.-C. Rousset. CARIN: A Representation Language Combining Horn Rules and Description Logics. In *European Conf. on Artificial Intelligence*, pages 323–327, 1996.

[17] D. L. McGuinness and J. R. Wright. An Industrial Strength Description Logic-based Configurator Platform. *IEEE Intelligent Systems*, 13(4):69–77, July/August 1998.