Description logic reasoning using the PTTP approach

Zsolt Nagy, Gergely Lukácsy, Péter Szeredi
Budapest University of Technology and Economics
Department of Computer Science and Information Theory
{zsnagy, lukacsy, szeredi}@cs.bme.hu

Abstract

The goal of this paper is to present how the Prolog Technology Theorem Proving (PTTP) approach can be used for ABox-reasoning. This work presents an inference algorithm over the language $\mathcal{ALC}$, and evaluates its performance highlighting the advantages and drawbacks of this method.

1 Introduction and motivation

Work reported in this paper is being carried out in the SINTAGMA[1] project, which aims at the development of a knowledge management tool-set for the integration of heterogenous information sources. This is an extension of the SILK[2] [2] technology, for retrieving information spanning over several data sources stored in the model warehouse of the system. The model warehouse contains UML models as well as models given using description logics (DL) [1].

Currently we are working on extending the capabilities of the SINTAGMA tool-set by designing and implementing description logic inference methods used for querying information sources containing large amounts of data. The first step of this research process resulted in a resolution-based transformation of ABox-reasoning problems to Prolog [10]. This algorithm is able to answer *instance-check* and *instance-retrieval* queries over the DL language $\mathcal{ALC}$ and an empty TBox. In this paper, we examine how ABox-reasoning services can be provided with respect to a *non-empty* TBox using Prolog technology.

This paper is structured as follows: Section 2 presents related work on ABox-inference in description logics. Section 3 details the method how ABox-reasoning

[1]Semantic INtegration Technology Applied in Grid-based Model-driven Architectures
[2]Semantic Integration via Logic and Knowledge

is performed in our framework. Section 4 evaluates the performance of our technique, highlighting its strong points and weaknesses. Finally, Section 5 concludes this work and addresses future research challenges.

2 Related Work

Traditional tableau-based Description Logic reasoners such Racer are slow when performing ABox-reasoning on large amounts of instances [4]. The work [7] describes a resolution-based inference algorithm, which is not as sensitive to the increase of the ABox size as a tableau-based method. The system Kaon2 [8] implements this method and provides reasoning services over the description logic language $\mathcal{SHIQ}$.

The work [5] discusses how a first order theorem prover such as Vampire can be modified and optimized for reasoning over description logic knowledge bases.

Paper [3] describes a direct transformation of $\mathcal{ALC}$ description logic axioms to Horn-clauses. Although [3] restricts the expressive power of $\mathcal{ALC}$ by disallowing constructs that result in non-Horn clauses, the main advantage of the approach is that the transformed clauses can be supplemented with other non-DL Horn-clauses.

Both the modified Vampire and Kaon2 aim to provide inference services over knowledge bases defined using the expressive power of the DL language in question plus the expressive power of a restricted fragment of first order logic. In case of Kaon2, the restrictions involving FOL components are such that the inference algorithm remains to be a decision procedure. It has been proved in [9] that query-answering over a knowledge base containing $\mathcal{SHOIN}$ axioms and so-called DL-safe rules is decidable. However in practice, this decision procedure has been shown to be highly inefficient due to don't know nondeterminism (backtrack search). For efficiency reasons, only a subset of the $\mathcal{SHOIN}$ description logic language is used in Kaon2.

In our previous work [10], we have provided a possible resolution-based alternative to ABox reasoning over the language $\mathcal{ALC}$ with respect to an empty TBox. In that approach, a query-plan was derived before the first ABox-access and the query-plan was executed using Prolog. This solution could be viewed as a two-phase proof of an ABox-query: first, the ABox-independent part of the proof is constructed, resulting in the query-plan as a Prolog program, and second, the query-plan is executed on the ABox.

Our current work deals with $\mathcal{ALC}$ ABox-reasoning in the presence of non-empty TBoxes, with some restrictions on the form of the TBox-axioms. However, we do allow full negation and disjunction on each side of the DL axioms, generalizing the transformation of [3]. In contrast with the earlier approach, *we delegate the whole reasoning process to Prolog*, building on the *Prolog Technology*

Theorem Proving (PTTP) approach [13].

Paper [6] introduces a fragment of the $\mathcal{SHIQ}$ language that can be transformed into Horn-clauses. The Horn-$\mathcal{SHIQ}$ language presented there allows more concept constructors than our restricted $\mathcal{ALC}$ framework. On the other hand, our approach poses less restrictions on use of disjunctions.

3 Transforming DL axioms to Prolog clauses

In this section, we describe the transformation of $\mathcal{ALC}$ TBox-axioms into executable Horn-clauses. The aim of this transformation is to provide *concept-instance check* and *concept-instance retrieval* services over an $\mathcal{ALC}$ knowledge-base containing both ABox- and TBox-axioms.

In order to avoid the appearance of Skolem-functions in the transformed Horn-clauses, some restrictions are posed on the DL-axioms. We exclude subsumption axioms $C \sqsubseteq D$ where $\forall R.E$ is a subconcept of the negational normal form of C or $\exists R.E$ is a subconcept of the negational normal form of D. Although the method described below can cope with ABox-inference on some TBoxes containing such axioms, allowing them in general may lead to non-termination when transformed to Prolog. The problem of termination for transformed clauses containing Skolem-functions may be addressed by reverting to a two-phase transformation process (as in [10]) and using ordered resolution [7] in the first phase, or by applying a proper meta-level cycle-detection technique.

The goal of this section is to show that an arbitrary DL knowledge-base obeying the above restriction can be transformed into a set of executable Prolog-clauses. There are three types of clauses: the TBox-clauses, ABox-facts and the clauses belonging to the instance-check and instance-retrieval queries. Currently, we use an interpreter written in Prolog for executing these Prolog clauses, in order to ease the development process and experimentation. However, it is fairly easy to transform these clauses further to code directly executable on a Prolog system.

Description of the transformation. Let an ABox $\mathcal{A}$ and a TBox $\mathcal{T}$ be given, where $\mathcal{T}$ consists of axioms $C \sqsubseteq D$, where C and D are in negational normal form, and C does not contain subconcepts of form $\forall R.E$, while D does not have a subconcept $\exists R.E$. The axioms of the TBox are transformed into Prolog clauses using the transformation steps below.

1. Based on the well-known mapping described e.g. in [1], we transform the description logic axioms into first order logic formulas.

2. The formulas corresponding to the TBox-axioms are then transformed into clausal form [12]. According to the properties of clause transformation, the generated clauses have the following properties:

- clauses are disjunctions of possibly negated literals;
- all variables in the clauses are universally quantified.

Due to our restrictions posed, no Skolem functions appear in the clausal form of the concepts. The general form of a transformed TBox-clause is thus the following:

$$\bigvee_m C_m(x_{i_m}) \vee \bigvee_n \neg D_n(x_{j_n}) \vee \bigvee_p \neg R_p(x_{k_p}, x_{l_p}), \tag{1}$$

where the literals C_m and D_n correspond to atomic concepts, and literals R_p correspond to role names, while x-es denote variables. Note that while both positive and negative unary literals can appear in the clauses, binary literals are only negative. Positive binary literals do not appear in any clause, since this would correspond to a role negation in the corresponding DL axiom.

3. Each TBox-clause

$$L_1 \vee L_2 \vee \ldots \vee L_n \tag{2}$$

is transformed into n clauses of the following form $(i = 1, \ldots, n)$

$$L_i \leftarrow L_1^{neg} \wedge L_2^{neg} \wedge \ldots \wedge L_{i-1}^{neg} \wedge L_{i+1}^{neg} \wedge \ldots \wedge L_n^{neg}, \tag{3}$$

where L_i is the head of the clause, and all other literals are body literals. L^{neg} is equal to $\neg L$ if L is a positive literal, and $L^{neg} = T$ if $L = \neg T$. A clause of form (3) is called a *contrapositive* of the clause (2).

Execution in Prolog. The resulting contrapositives are then transformed to Prolog syntax and are executed by our interpreter. The transformation and interpretation techniques as well as the usage of contrapositives have been borrowed from PTTP (Prolog Technology Theorem Prover) [13]. We briefly describe how we handle these issues:

- *Occurence of positive and negative literals*: To transform an arbitrary clause to Prolog format we introduce new predicate names. For each concept-name C (i.e. one of C_m or D_n of Formula (1)) we add a new predicate name *nonC*, and replace all occurrences of $\neg C(X)$ by $nonC(X)$ both in the head and in the body. The link between the separate predicates C and *nonC* is created by ancestor resolution, see below.

- *Ancestor resolution*: *open* predicate calls[3] are collected in an *ancestor list.* If the ancestor list contains a literal which can be unified with the first

[3]I.e. calls which were entered or re-entered, but have not been exited yet, according to the Procedure-Box model of Prolog execution. [11]

literal of the goal, then the call corresponding to the goal literal succeeds. Program execution is divided into two branches: one reflecting the modifications caused by the ancestor resolution step, and one without using ancestor resolution.

- *Loop elimination*: if the first literal of the goal can be found in the ancestor list, we stop the given branch with a failure. The term 'can be found' is interpreted by the `==` Prolog built-in predicate, which succeeds if its operands are identical.

Prolog execution uses SLD-resolution [11], which is a linear resolution strategy always resolving the first literal of the *goal* with the head of a corresponding definite-clause in the Prolog-program. When all n contrapositives of a clause (2) are available, the goal can be resolved with *any* literal L_i of the clause. Thus, any linear refutation can be simulated in Prolog at the expense of introducing multiple variants of the clauses.

Soundness, completeness, termination. Soundness and completeness of this approach is based on the properties of the PTTP technique [13], since PTTP is a sound and complete first order theorem prover. A DL ABox-inference problem is handled with these theorem-proving techniques by resolving the set of produced TBox-, ABox- and query-clauses in a PTTP framework using Prolog.

Regarding termination, consider the following example: if we transform the axiom $C \sqsubseteq \forall R.C$[4] into clausal form, we get the clause $\neg C(x) \vee \neg R(x, y) \vee C(y)$. Executing an instance-retrieval query on concept C may lead to an infinite loop if the literals in the contrapositives are not ordered properly. Whenever we call the contrapositive

```
C(Y) :- C(X), R(X, Y).
```

we introduce a new variable inside the literal `C`, which is not detected by the loop elimination technique. This example shows that without further provisions, the termination of the Prolog program is not guaranteed.

Conjecture: *The execution of the resulting clauses of the above described transformation always terminates if all role literals in the body of the clauses are moved before concept literals.* This rearrangement ensures that at the time of a concept-predicate call the variable in the call either (a) occurs in a role literal previously called or (b) is equal to the head-argument of the clause. Role literals can only be resolved with ABox-facts, so all variables occurring in a role literal are unified with ABox-instances. In case (a), the variable in the concept-predicate call is instantiated due to the previous role-predicate call. In case (b), the head-argument of the clause is either instantiated, or is identical to an

[4]Suppose that C is an atomic concept.

older head argument. By induction, this means that any uninstantiated concept argument must be identical to the parameter of the instance-retrieval query. The number of clauses and instances is finite and only one variable, namely the parameter of the query may occur in a predicate call. Therefore, for a set of clauses, only a finite number of different predicate calls is possible. Since loop elimination ensures that the same predicate call never executes twice, execution always terminates.

Optimizations. Note that not all contrapositives of the clauses are needed for acquiring a complete decision procedure. Contrapositives containing a role literal in the head can be omitted, since they cannot be called within the execution of an instance-check or an instance-retrieval query. Although these clauses are never called by the program, the size of the program gets smaller, reducing administration overhead.

It is often the case that a concept predicate is called with a ground argument. In the presence of disjunctions, such a Prolog goal could be executed in multiple ways. Obviously, once the goal exited successfully, there is no point in exploring alternative branches within this goal. Therefore we modified the interpreter to perform a Prolog cut operation (`!`) after a successful exit from a ground goal. This optimization resulted in a measurable performance boost.

Composite queries. The outcome of the transformation is a set of Prolog-clauses, which are usable for instance-check or instance-retrieval queries on possibly negated atomic concepts. For allowing ABox-inference queries containing a composite query-concept Q, an atomic concept-name A has to be introduced for the query-concept with the axiom $A \equiv Q$. The equivalence-axiom $A \equiv Q$ can be written in form $A \sqsubseteq Q$ and $Q \sqsubseteq A$. From these two subsumption axioms, only the second one has to be added to the TBox for answering the ABox query. This axiom is transformed to Prolog clauses in the same way as the other axioms in the TBox.

Note that directly transforming the composite query-concept Q to a Prolog query is not sufficient, because all the contrapositives of the query-clause $Q \sqsubseteq A$ may be needed for handling case-analysis. This technique is an alternative for passing the parameter of the instance-check and the instance-retrieval queries when handling case-analysis, as detailed in [10].

4 Performance evaluation

We have carried out the preliminary performance-evaluation of our approach and compared it with the available state-of-the-art description logic reasoners, such as Racer and Kaon2. Since the capabilities and usage of these systems are different, the goal of this comparison is to analyze the behavior of these approaches for different reasoning tasks and not to declare a winner among these systems. It is important to note that these inference engines provide reasoning services over different description logic languages, so the comparison has to be evaluated with care.

The tests were run on an AMD Athlon CPU running at 1.81GHz with 1GB RAM and Windows XP operating system with Service Pack 2. After a test had been performed, RACER[5], KAON2[6] and our Prolog interpreter[7] were always reinitialized.

We show the results for two instance-retrieval test cases. The first test (the first five-line block of Table 1) summarizes the results of a test-case over a TBox containing TBox-axioms with no disjunctions. The second test was run on a TBox containing complex TBox-axioms with disjunctions and composite concepts. The last line belongs to the second test case indicating a special case explained below. The ABox of the tests was randomly generated using a Prolog program.

KB characteristics	# instances	#role assertions	#concept assertions	RACER	KAON2	SICSTUS
simple	500	1269	337	1.1s	0.203s	0.000s
TBox	1000	2499	688	3.1s	0.531s	0.000s
axioms	2000	2133	1333	13.2s	0.515s	0.016s
	5000	12599	3300	91.4s	0.468s	0.016s
	10000	25036	6627	>300s	0.891s	0.031s
complex	500	261	323	0.0s	0.078s	0.890s
TBox	1000	503	635	0.7s	0.375s	1.953s
axioms	2000	972	1392	1.5s	1.062s	7.875s
	5000	2493	3367	7.1s	5.156s	44.265s
Special complex	5000	2493	3367	7.0s	3.579s	0.813s

Table 1: Test results

The first block shows that the resolution-based approaches are not as sensitive to the increase of the size of the ABox as the tableau-based RACER, when not many disjunctions are present[8]. When comparing execution times of KAON2 and our interpreter, one should note that KAON2 provides inference services over a description logic language with higher expressive power.

[5]version 1.7.6

[6]version released on 8th December 2005

[7]running under SICStus Prolog version 3.12.2

[8]Note that in special cases as shown in [10], KAON2 fails to scale.

The second test case highlights that our interpreter does not handle disjunctions efficiently enough, due to the many introduced contrapositives that often make the program explore irrelevant parts of the search space. For instance, if we know that some TBox-axioms are not needed for answering a simple query, the clauses belonging to the axioms not needed can be left out, reducing execution time below one second for even the ABox with size 5000. This phenomenon is described as a special test case in the last row of Table 1. Here the axioms not needed for answering the instance-retrieval query are omitted from the TBox. Note that for TBoxes containing less disjunctions, the execution time was only reduced by a small constant factor when omitting irrelevant axioms. KAON2 seems to handle disjunctions fairly well. Although RACER was faster than KAON2 for the first test case, execution times increased with the overhead of examining all the instances of larger ABoxes.

A possible reason why our approach does not handle disjunctions well enough is that – as opposed to our previous work [10] – no preprocessing is made on the transformed PTTP-clauses. We believe that the current approach can be optimized by cutting the search space at compile time.

5 Conclusion and future work

This paper has described an ABox-reasoning technique for the DL language $\mathcal{ALC}$ which is able to handle TBoxes obeying certain restrictions. We have written an interpreter which executes the resolution-based proof belonging to instance-check and instance-retrieval queries and have also evaluated the performance of this approach, pointing out its advantages and disadvantages.

In the future, our plan is to examine how the services provided by the interpreter can be included in the Prolog-program serving as the query-plan. Our other aim is to apply optimizations guiding the Prolog execution of the refutation. By combining this approach with the approach seen in our previous work, we believe that the performance of this reasoning technique can be largely enhanced. A method handling all forms of refutation involving Skolem-functions and DL-constructs not addressed in this paper is also subject of future research.

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, 2003.

[2] Tamás Benkő, Gergely Lukácsy, Attila Fokt, Péter Szeredi, Imre Kilián, and Péter Krauth. Information integration through reasoning on meta-data. In *Proceedings of the Workshop "AI Moves*

to IA", IJCAI'03, Acapulco, Mexico, pages 65–77, August 2003. http://www.dimi.uniud.it/workshop/ai2ia/cameraready/benko.pdf.

[3] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description Logic Programs: Combining Logic Programs with Description Logics. In *Proc. of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 48–57. ACM, 2003.

[4] 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.

[5] Ian Horrocks and Andrei Voronkov. Reasoning support for expressive ontology languages using a theorem prover. In *FoIKS*, volume 3861 of *Lecture Notes in Computer Science*, pages 201–218. Springer, 2006.

[6] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Data complexity of reasoning in very expressive description logics. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 466–471. International Joint Conferences on Artificial Intelligence, 2005.

[7] U. Hustadt, B. Motik, and U. Sattler. Reasoning for description logics around $\mathcal{SHIQ}$ in a resolution framework. Technical Report 3-8-04/04, FZI, Karlsruhe, Germany, June 2004.

[8] KAON2: Ontology management tool for the semantic web. http://kaon2.semanticweb.org/.

[9] Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for OWL-DL with rules. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60, July 2005.

[10] Zsolt Nagy, Gergely Lukácsy, and Péter Szeredi. Translating description logic queries to Prolog. In *PADL*, volume 3819 of *Lecture Notes in Computer Science*, pages 168–182, 2006.

[11] U. Nilsson and J. Maluszynski, editors. *Logic, Programming and Prolog.* John Wiley and Sons Ltd., 1990.

[12] S.J. Russel and P. Norvig, editors. *Artificial intelligence: a modern approach.* Prentice Hall, 1994.

[13] M. Stickel. A Prolog technology theorem prover: A new exposition and implementation in Prolog, Technical Note 464, SRI international, 1989.