Prolog Execution Engines for Description Logic Reasoners

Gergely Lukácsy, Zsolt Nagy, Péter Szeredi

Budapest University of Technology and Economics
Department of Computer Science and Information Theory
{lukacsy, zsnagy, szeredi}@cs.bme.hu

Introduction. The goal of this poster is to present some Prolog-based [3] alternatives for Description Logic ABox-reasoning. We believe that Prolog can be used well for extending DL formalisms with rules. The top-down executional mechanism of Prolog suits the ABox-reasoning task by ignoring irrelevant data, and it is also worth to mention that the Prolog community has been developping Prolog for more than 30 years, providing a highly optimized logic programming environment. In our approaches, both the ABox-reasoning code and the attached rules can be executed in a Prolog framework. Queries are answered using the PTTP (Prolog Technology Theorem Proving) [4] approach. The aim of these reasoning algorithms is to efficiently answer instance-check and instance-retrieval queries when sizeable amounts of data are stored in the ABox. We describe two approaches of transforming a DL knowledge-base to Prolog clauses.

Soundness and completeness of both approaches are based on resolution and the PTTP technique. In the implementation of the ideas, procedural elements of Prolog such as the cut for filtering multiple solutions out are used for optimizing the code.

The Restricted Approach. [2] This approach provides ABox-reasoning services over an empty TBox. Let an extensionally reduced ABox $\mathcal{A}$ be given with the property that $\mathcal{A}$ is satisfiable[1]. The goal is to determine all the instances of an $\mathcal{ALC}$ query-concept C, or as a special case, determine if an individual o is an instance of the query-concept C. Reasoning is split into two parts: first a Prolog execution plan is produced, then the plan is executed on the ABox.

More specifically, we first transform the $\mathcal{ALC}$ query-concept into a union of tree-concepts. A *tree-concept* is an $\mathcal{ALC}$ concept formed using the intersection, existential restriction and atomic negation constructors only. Each tree-concept is transformed into a piece of Prolog-code individually. Prolog clauses belonging to each tree-concept have to be executed in a common namespace. We refer the reader to [2] on the details of the transformation.

The Intermediate Approach. [1] This approach provides ABox reasoning services over an extensionally reduced $\mathcal{ALC}$ knowledge-base containing an ABox and a restricted TBox. We exclude subsumption axioms $C \sqsubseteq D$ from the TBox

[1] We do not deal with ABoxes containing contradictions. When reading the assertions, we do not allow both $A(i)$ and $\neg A(i)$ to be simultaneously asserted for any A or i.

where $\forall R.E$ is a subconcept of the negation normal form of C or $\exists R.E$ is a subconcept of the negation normal form of D. This restriction is due to the fact that the current reasoning algorithm cannot handle Horn-clauses containing Skolem-functions. Let a TBox $\mathcal{T}$ conforming to the restrictions above and an extensionally reduced ABox $\mathcal{A}$ be given. The content of $\mathcal{T}$ is transformed to Prolog rules and the content of $\mathcal{A}$ is transformed to Prolog facts.

Comparison of the approaches. We have designed and experimented with a case-study on fault-tolerant behavior of systems. Using this case study, we compared the performance of the two approaches.

We summarize the basic differences between the two approaches in Table 1.

The *Restricted* approach presented in [2] focuses on ABox-reasoning over an empty TBox. Here, it is also possible to add non-DL Horn-clauses to the transformed set of clauses. These Horn-clauses make it possible for the knowledge-engineer to describe terminological knowledge regarding the instances of the knowledge-base. The main advantage of this approach is its performance and scalability.

On the other hand, the *Intermediate* approach [1] is able to provide ABox-inference services over a non-empty terminology box. The main advantage of this technique is that it provides reasoning services over a knowledge-base containing (a) a slightly restricted $\mathcal{ALC}$ TBox, (b) terminology level knowledge represented using Horn-clauses and (c) ABox-instances and relations. The content of the knowledge-base is transformed into Horn-clauses which are executed in Prolog using the PTTP technique. Execution is currently done by using our own PTTP Horn-clause interpreter. Preprocessing optimizations do not appear in this work, the solution is derived from the PTTP inference engine only. Although this approach is capable of solving ABox-inference problems over a non-empty TBox, we still refer to it as the *Intermediate Approach*, since we plan getting rid of the restrictions involving the TBox and the interpreted execution.

Table 1. Comparison of the two reasoning approaches.

Approach	*Restricted*	*Intermediate*
Expressive power	$\mathcal{ALC}$ ABox, no TBox	$\mathcal{ALC}$ ABox and restricted TBox
Preprocessing	all ABox-independent steps	none
Execution plan	runnable Prolog program	interpreted PTTP clauses

Summary, future work. We believe that Prolog-based ABox-reasoning can be well combined with Prolog rules and we found or initial results encouraging. In the future, we would like to combine the advantages of the two approaches and form an approach capable of efficiently handling ABox-reasoning over an arbitrary TBox and ABox.

References

1. Zsolt Nagy, Gergely Lukácsy, and Péter Szeredi. Description logic reasoning using the PTTP approach. to appear in the proceedings of dl2006. international description logic workshop, windermere, england., 2006.
2. 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.
3. U. Nilsson and J. Maluszynski, editors. *Logic, Programming and Prolog.* John Wiley and Sons Ltd., 1990.
4. M. Stickel. A Prolog technology theorem prover: A new exposition and implementation in Prolog, Technical Note 464, SRI international, 1989.