DL Reasoner vs. First-Order Prover

Dmitry Tsarkov and Ian Horrocks
Department of Computer Science
The University of Manchester
Manchester, UK
{tsarkov|horrocks}@cs.man.ac.uk

Abstract

We compare the performance of a DL reasoner with a FO prover on reasoning problems encountered during the classification of realistic knowledge bases.

1 Introduction

Description logics (DLs) can, in general, be viewed as decidable subsets of first-order logic (FOL). It is, therefore, natural to compare the performance of dedicated DL reasoners with general purpose FOL theorem provers. On the one hand, using available FOL provers could, potentially, save a great deal of effort in developing algorithms and reasoners for various DLs. On the other hand, FOL provers could be used to provide reasoning services for expressive DLs for which specific algorithms have yet to be developed.

Hustadt and Schmidt [8] have investigated the use of FOL provers to reason with propositional modal logics, and, via well known correspondences [12], with description logics. They used the SPASS FOL prover, and a relatively complex functional translation which produces a subset of FOL for which SPASS is able to guarantee complete reasoning. The results of this experiment were quite encouraging, with performance of the SPASS based system being comparable, in many cases, with that of state of the art DL reasoners. The tests, however, mainly concentrated on checking the satisfiability of (large) single modal logic formulae (equivalently, DL concepts), rather than the more interesting (in a DL context) task of checking the satisfiability of formulae w.r.t. a large theory (equivalently, a DL knowledge base). Moreover, this approach has yet to be extended to the more expressive DLs that underpin ontology languages such as OIL, DAML+OIL and OWL [7].

An alternative approach, and the one we investigate here, is to use a simple "direct" translation based on the standard first order semantics of DLs (see, e.g., [1]).[1] Using this approach, a TBox, given as set of DL axioms, is translated into a FO theory, defined as a set of FO axioms. Then a DL reasoning task (w.r.t. the TBox) is transformed into a FO task that uses the theory. The translation ϕ maps DL concept and role names into unary and binary predicates respectively. Complex concepts are mapped into formulae as shown below. DL axioms are translated into FO axioms in the obvious way. For example, subsumption and equivalence axioms are translated into, respectively, FO implication and equality axioms (with the free variables universally quantified).

[1] The resulting theories will be equisatisfiable w.r.t. formulae produced by Hustadt and Schmidt's functional translation, but may lead to less efficient (and incomplete) reasoning—at least for the SPASS prover.

As an example, let's see a translation of a couple of concept and role axioms:

DL	FOL
$R \sqsubseteq S$	$\forall x \forall y(\phi_R(x,y) \rightarrow \phi_S(x,y))$
$C \equiv D \sqcap \exists R.(E \sqcup$	$\forall x(\phi_C(x) = \phi_D(x) \wedge \exists y(\phi_R(x,y) \wedge (\phi_E(y) \vee$
$\forall S^{-}.F)$	$\forall x(\phi_S(x,y) \wedge \phi_F(x)))))$
$A \sqsubseteq \geqslant 3\, R.B$	$\forall x(\phi_A(x) \rightarrow \exists y_1 \exists y_2 \exists y_3(\phi_R(x,y_1) \wedge \phi_B(y_1) \wedge$
	$\phi_R(x,y_2) \wedge \phi_B(y_2) \wedge \phi_R(x,y_3) \wedge \phi_B(y_3) \wedge$
	$(y_1 \neq y_2) \wedge (y_2 \neq y_3) \wedge (y_1 \neq y_3))$
$Trans(T)$	$\forall x \forall y \forall z(\phi_T(x,y) \wedge \phi_T(y,z) \rightarrow \phi_T(x,z))$

For simple DLs (like $\mathcal{ALC}$) the translation is into the FOL class $\mathcal{L}^2$ (the FOL fragment with no function symbols and only 2 variables), which is known to be decidable.[2] The above translations of the role inclusion axiom and concept equality axiom are, for example, in $\mathcal{L}^2$. When number restrictions are added to these DLs, they can be translated into $\mathcal{C}^2$—equivalent to $\mathcal{L}^2$ with additional "counting quantifiers"—which is also known to be decidable [1]. Most implemented FOL provers do not, however, support counting quantifiers, so in the above example the number restriction has been translated into FOL with equality using more than two variables. This translation is obviously problematical when a very large number is used in a number restriction, as the number of inequalities increases with the square of the number used in the restriction. Such cases are, however, extremely rare in realistic ontologies.

More expressive description logics, e.g., with transitive roles ($\mathcal{SHIQ}$) and/or complex role axioms ($\mathcal{RIQ}$), need at least three variables for their First-Order translation. The above transitivity axiom for role T is an example of this case. FOL with three variables is known to be undecidable [3]. Such description logics are now widely used, e.g., in large KBs describing complex domains such as medicine [10], and as the basis for Semantic Web ontology languages such as DAML+OIL and OWL.

For this reason, we have decided to investigate the use of the above straightforward translation method with a state of the art FO theorem prover. In the last few years, a number of highly efficient FO provers have been implemented [9, 13, 11]. This provers compete annually on a set of tasks, and the results are published [4]. The best general-purposes prover in recent years has been Vampire [11], and we have chosen this prover to use in our comparison.

2 Preliminaries

Like most efficient state-of-the-art automated theorem provers (ATPs), Vampire implements saturation with resolution and paramodulation. The essence of the approach is as follows. The input FO task is given as a set of *clauses*. Each clause is a disjunction of *literals*. The prover then tries to deduce new clauses from the initial set, using the *resolution* or *paramodulation* rules. This goes on until an empty clause (i.e., a disjunction of zero size) is inferred, or it is not possible to create any new clauses. If the empty clause was inferred, then the task (initial set of clauses) is unsatisfiable. This approach is *sound*, i.e., the empty clause will only be inferred, if the FO task is unsatisfiable.

Vampire, like most other general-purposes ATPs, uses various strategies. Some of them are *complete*. This means that if it is impossible to create a new clause from the existing set

[2] In order to stay in $\mathcal{L}^2$ the same two variables need to be used in alternating order in nested subformulae—see [1].

of clauses, then the input task is satisfiable. Such set is called *saturated*. There are also some *incomplete* strategies; they are used for faster search in some special cases. Even using a complete strategy, it is not always possible to saturate the initial set: saturation does not always converge, and even if it does it may use too much time on realistic problems. This means that saturation-based provers such as Vampire are usually quite effective on unsatisfiable problems (because they can often derive the empty clause before exhausting their resources), but much less effective on satisfiable problems (because they may simply continue to generate new clauses until their resources are exhausted). Note when a subsumption holds, the corresponding FO problem will be unsatisfiable, so ATPs are relatively good at proving subsumption (positive subsumption), and much less good at proving non-subsumption (negative subsumption).

In view of the relative unpredictability of their behaviour, testing an ATP is usually organized as follows. The ATP takes an input task with a time limit. It uses some (complete) strategy which tries to infer an empty clause. Three answers are allowed: ”satisfiable”, ”unsatisfiable” and ”timeout”.

Key DL reasoning tasks (for the TBox) include KB consistency, concept satisfiability, concept subsumption and classification. All of these tasks are reducible to KB consistency or, equivalently, concept subsumption w.r.t. a KB (in fact they are reducible to concept satisfiability using internalisation). The classification task transforms into a (quite large) set of subsumption tasks.

As a standard DL reasoner we will use FaCT++ version 0.8. This system is a next generation of the well-known FaCT reasoner [6], being developed as part of the EU WonderWeb project (see `http://wonderweb.semanticweb.org/`). This implementation is based on the same Tableaux algorithms as the original FaCT, but has a different architecture and is written in C++ instead of LISP. the goal of developing FaCT++ was to create a modern reasoner for complex DLs (like $\mathcal{SHIQ}$, $\mathcal{OWL}$) with good performance, high extendibility and internal data structures that are better able to handle very large knowledge bases. FaCT++ is not as efficient as FaCT yet, because not all possible optimisations have been implemented, but it has comparable performance characteristics. FaCT++ has a customisable architecture, so it is possible to implement different tactics for reasoning and choose the best one for a given TBox.

3 Basic comparison

3.1 Experimental methodology

We used the following comparison scheme. For a given KB, the DL reasoner performed the classification process. In the case where a subsumption test is necessary, a task was also generated for the FO prover, and the result of the DL testing was noted for future comparison. After finishing the classification process, the set of problems that were solved had been generated. The FO prover was then run on this set of problems. The results were recorded and then compared with the DL ones. Note that the time taken in generating the FO problem and converting it to clausal normal form (CNF) is not considered in the experiment, as in a realistic system the FO problems would be generated in CNF and would be communicated directly to the FO prover rather than being written to a file.

We ran Vampire with a fixed time limit: 300 seconds per task. A larger time limit could, of course, lead to solving more tasks. The other parameters were standard, except setting pre-

TBox	Concepts	Roles	Pos subs	Neg subs
Tambis	345	107	96	2241
Platt	313	14	140	17286
Galen	2749	207	4042	57091

Table 1: Knowledge bases used for the comparison

processing to minimum level (with option "`--elim-def 0`") and setting resolution strategy to hyperresolution (with option "`--selection 2`"). Empirically, it was discovered that preprocessing is both time consuming and ineffective, and that hyperresolution was the most effective strategy available. Vampire is highly tunable, and even better performance may be achievable by adjusting other parameters.

Because of the number of tasks generated when classifying a large KB, and the nature of Vampire's behavior (troubles with saturating large sets of clauses in the case of failed subsumption tests), not all problems generated by FaCT++ were tested using Vampire. We used all positive subsumption tests, and a small number (between 1% and 10%) of negative subsumption tests for each KB. The negative tests were chosen randomly from the entire set.

Note that actual number of comparisons depends on the DL systems' classification algorithm. If the algorithm is optimised (i.e. based on concept structure, results of previous operations, etc), then the number of (logical) subsumption tests may be much smaller than using a naive (unoptimised) algorithm. In the FaCT++ classifier, the number of actually performed subsumption tests is typically only 2-5% of those that would be required in a "brute force" implementation (see [2]). Relatively few of these tests give a positive result, because most subsumptions are "obvious", and do not need to be computed using a logical subsumption test.

3.2 Test models

We used three KBs in the test; their parameters are given in the Table 1. The Tambis KB has a small size (345 concepts, 107 roles) and quite simple structure. The Platt KB has a similar size (313 concepts, 14 roles), but a more complicated structure and 13 general concept inclusion axioms (GCIs) [1]. The Galen KB has very large size (2749 concepts, 207 roles) with quite a simple concept language (only conjunction and existential restrictions), but with transitive roles and a large number of GCIs. The comparison used all positive subsumption problems plus 10% of Tambis' negative subsumption problems, 1% of Platt's negative subsumption problems and 1% of Galen's negative subsumption problems.

Results using the basic translation may be viewed in table 2. Columns *Pos* and *Neg* give the number of positive and negative tasks; *time* is the average time (in seconds) per solved task; *S. Pos* and *S. Neg* give the number of positive and negative tasks solved.

The only task Vampire cannot solve in the Platt KB is a negative subsumption problem (it finishes with an "Out of time" result). All other tasks in the Platt and Tambis tests were solved in times from 0 to 1.4 sec per task. In all cases a complete proving strategy was used.

For comparison purposes, we ran FaCT++ with its absorption optimisation disabled (see [1]). In this configuration, FaCT++ cannot classify the largest KB (Galen) due to memory limits. The reason for this is the large number of GCIs). Creating FO tasks for Vampire lead to the solution of many of the chosen tasks, illustrating that Vampire with hyperresolution is quite efficient, even with the (heavy) usage of GCIs.

tool	KB	Pos	Neg	time	S. Pos	S. Neg
FaCT++	Tambis	96	2241	0	96	2241
Vampire	Tambis	96	227	0.2	96	227
FaCT++	Platt	140	17286	0	140	17286
Vampire	Platt	140	173	0.55	140	172
FaCT++	Galen	4042	57091	—	0	0
Vampire	Galen	4042	574	82.6	3958	373

Table 2: Comparison results for full-sized translation

tool	KB	Pos	Neg	time	S. Pos	S. Neg
FaCT++	Tambis	96	2241	0	96	2241
Vampire	Tambis	96	227	0.2	96	227
FaCT++	Platt	140	17286	0	140	17286
Vampire	Platt	140	173	1.09	140	172
FaCT++	Galen	4042	57091	0.01	4042	57091
Vampire	Galen	4042	574	66.04	3963	375

Table 3: Comparison results for absorbed translation

4 Axiom absorption

One of the main factors that adversely affects performance is the presence of GCIs. With tableaux algorithms, every GCI must must be satisfied in every tableaux state. This can lead to an astronomical amount of calculation for the 408 GCIs of the Galen KB. Similarly, GCIs lead to a great number of extra FO steps that usually slow down the theorem prover. Moreover, a number of FO optimisations cannot work properly with GCIs.

It is sometimes possible to reduce global axioms to additional concept implications. This process is called *absorption* [5]. This optimisation is equally applicable to the FO case, so we have also tested the FO reasoner using tasks that are created *after* the DL reasoner has pre-absorbed global axioms. This leads to smaller and more convenient (for reasoning) FO axiom sets. All global axioms from the Platt and galen KBs were removed by the absorption process. Results using the pre-absorbed translation may be viewed in table 3.

Absorption greatly improves the performance of FaCT++, and the Galen KB was completely solved in quite small time. In FO proving, however, we see quite different situation: the Platt KB tests took two times longer, whilst Galen KB tests were 20% faster.

5 Relevant-only translation

Absorption has little affect on the performance of Vampire because its saturation algorithm runs “globally”, i.e., the nature of resolution algorithm leads to the use of all the available information. In contrast, the tableaux algorithm used in FaCT++ runs “locally”, i.e., it ignores a lot of information that is unnecessary for proving a given subsumption task.

An obvious way to correct this situation is to remove all unnecessary information from the FO task. This will lead to a small overhead in computational complexity, because the DL reasoning process does not need to apply this procedure. On the other hand, this operation should greatly increase the performance of the FO prover.

For computing a subsumption $C \sqsubseteq D$, the “relevant” information is that which is relevant to concept C or to concept D. In order to check subsumption w.r.t. a TBox, it is also necessary

FindRelevant (conceptExpression E)
1. **forall** $[CN \in E]$ **do** MarkConceptRelevant (CN); **end do**
2. **forall** $[RN \in E]$ **do** MarkRoleRelevant (RN); **end do**

MarkConceptRelevant (concept C)
1. **if** $(C.Relevant = true)$ **then return**;
2. **else**
3. $\quad C.Relevant := true$;
4. $\quad$ **forall** [$C \sqsubseteq D$ or $C \equiv D$ in KB] **do** FindRelevant(D); **end do**
5. **end if**

MarkRoleRelevant (role R)
1. **if** $(R.Relevant = true)$ **then return**;
2. **else**
3. $\quad R.Relevant := true$;
4. $\quad$ **forall** $[RN : R \sqsubseteq RN]$ **do** MarkRoleRelevant (RN); **end do**
5. **end if**

Figure 1: Algorithms for locating relevant concepts and roles

tool	KB	Pos	Neg	time	S. Pos	S. Neg
FaCT++	Tambis	96	2241	0	96	2241
Vampire	Tambis	96	227	0	96	227
FaCT++	Platt	140	17286	0	140	17286
Vampire	Platt	140	173	0.01	140	173
FaCT++	Galen	4042	57091	0.01	4042	57091
Vampire	Galen	404	574	1.15	399	381

Table 4: Comparison results for absorbed minimal-sized translation

add information relevant to all the GCIs in the TBox.

The process of selecting information relevant to a concept expression E looks very much the same as *unfolding* (see [1]), and assumes that the KB is separated into a set of unfoldable axioms and a set of GCIs. Every concept name CN and role name RN appearing in E is relevant to E. The process is then repeated for unfoldable axioms with CN on the left hand side (whether inclusion or equality axioms). Also, if role R is relevant to E, then so are all it's super-roles $R' : R \sqsubseteq R'$, along with their inverses (if the target DL allows inverse roles).

An algorithm for computing relevant information is described in Fig. 1. It should be run for concepts in the subsumption test plus concepts occurring in GCIs of the TBox (if any). After applying the algorithm, only unfoldable axioms having concepts and roles with a raised $Relevant$ flag on their left hand side should be translated to the FO task description.

Results using the relevant-only translation may be viewed in table 4. Note that here only the Vampire running times are changed. In the current experimental implementation there is a significant overhead (comparable to the DL subsumption testing time) for creating a relevant-only task. This time is not, however, included in the table, as it would clearly be possible to implement the relevant-only translation *much* more efficiently. In order to keep the time required to carry out the experiment within reasonable bounds, we also reduced the number of positive tests for the Galen KB, taking 10% at random.

The removal of irrelevant information results in a great improvement for the Vampire

reasoner. In the small tasks (Tambis and Platt) Vampire reasoning takes practically the same time as FaCT++, both for positive and negative tests. For the Galen KB, the number of solved tests increased, and the average time was much smaller.

6 Discussion

As can be seen from the results, the performance of Vampire is much worse than that of FaCT++ when tested with reasoning tasks derived from a a naive translation of subsumption tests (w.r.t. a KB). When a suitably optimised translation is used, however, the performance of Vampire improves dramatically: for smaller KBs it is comparable with that of FaCT++, although for the Galen KB it is still in the order of 100 times slower. Vampire is able to solve the vast majority of tasks within the 300s time limit, but for the Galen KB it still fails on approximately 1% of positive tests (subsumption) and 34% of negative tests (non-subsumption). Unfortunately, the vast majority of test performed during KB classification are negative tests. The number of negative tests to be performed could, however, be significantly reduced by using caching and model-merging optimisations which are, as yet, not implemented in FaCT++. It should also be pointed out that performing each subsumption test in isolation puts Vampire at a considerable disadvantage, as fixed startup costs are incurred in every test, and information from previous tests cannot be reused.

The performance of Vampire is sufficiently encouraging to suggest that further investigations of FO theorem proving techniques would be worthwhile. The FO tasks generated in the tests are in the TPTP format [4], which is a de-facto standard for the theorem proving community, making it easy to use other FO provers in a similar comparison. Given that the performance of FO provers can vary greatly depending on the type of problem, it may be that another FO prover would give better performance on DL subsumption reasoning tasks.

Although the results presented here do not suggest that FO provers might be used to replace dedicated DL reasoners, it could still be useful to use a FO prover in a hybrid tool for dealing with complex DLs. For some very expressive DLs (like OWL), there is no known DL algorithm for reasoning with the full language (e.g., when nominals and inverse roles are used together). In this case, it would be possible to use a FO prover to compute some or all of the relevant subsumption tests. Although there would inevitably be problems with the speed of response, and with incompleteness, there would still be in improvement in performance given that DL reasoners currently can't deal with this situation *at all*.

References

[1] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2002.

[2] Franz Baader, Bernhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, and Enrico Franconi. An empirical analysis of optimization techniques for terminological representation systems. In *Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'92)*, pages 270–281. Morgan Kaufmann, Los Altos, 1992.

[3] Alexander Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1–2):353–367, 1996.

[4] CaSC ATP Competition. http://www.cs.miami.edu/∼tptp/casc/.

[5] I. Horrocks and S. Tobies. Reasoning with axioms: Theory and practice. In *Proc. of the 7th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'2000)*, pages 285–296, 2000.

[6] Ian Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.

[7] Ian Horrocks and Peter F. Patel-Schneider. The generation of DAML+OIL. In *Proc. of the 2001 Description Logic Workshop (DL 2001)*, pages 30–35. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-49/`, 2001.

[8] U. Hustadt and R. A. Schmidt. MSPASS: Modal reasoning by translation and first-order resolution. In R. Dyckhoff, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference (TABLEAUX 2000)*, volume 1847 of *Lecture Notes in Artificial Intelligence*, pages 67–71. Springer, 2000.

[9] Max Moser, Ortrun Ibens, Reinhold Letz, Joachim Steinbach, Christoph Goller, Johann Schumann, and Klaus Mayr. SETHEO and e-SETHEO - the CADE-13 systems. *Journal of Automated Reasoning*, 18(2):237–246, 1997.

[10] A. L. Rector, W. A. Nowlan, and A. Glowinski. Goals for concept representation in the GALEN project. In *Proc. of the 17th Annual Symposium on Computer Applications in Medical Care (SCAMC'93)*, pages 414–418, Washington DC, USA, 1993.

[11] A. Riazanov and A. Voronkov. The Design and Implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.

[12] Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 466–471, 1991.

[13] T. Tammet. Gandalf. *Journal of Automated Reasoning*, 18(2):199–204, 1997.