FaCT++ Description Logic Reasoner: System Description

Dmitry Tsarkov and Ian Horrocks

School of Computer Science
The University of Manchester
Manchester, UK
{tsarkov|horrocks}@cs.man.ac.uk

Abstract. This is a system description of the Description Logic reasoner FaCT++. The reasoner implements a tableaux decision procedure for the well known $\mathcal{SHOIQ}$ description logic, with additional support for datatypes, including strings and integers. The system employs a wide range of performance enhancing optimisations, including both standard techniques (such as absorption and model merging) and newly developed ones (such as ordering heuristics and taxonomic classification). FaCT++ can, via the standard DIG interface, be used to provide reasoning services for ontology engineering tools supporting the OWL DL ontology language.

1 Introduction

Description Logics (DLs) are a family of logic based knowledge representation formalisms [1]. Although they have a range of applications, they are perhaps best known as the basis for widely used ontology languages such as OIL, DAML+OIL and OWL [5].

A key motivation for basing ontology languages on DLs is that DL systems can then be used to provide computational services for ontology tools and applications [8, 9]. The increasing use of ontologies, along with increases in their size and complexity, brings with it a need for efficient DL reasoners. Given the high worst case complexity of the satisfiability/subsumption problem for the DLs in question (at least ExpTime-complete), optimisations that exploit the structure of typical ontologies are crucial to the viability of such reasoners.

FaCT++ is a new DL reasoner designed as a platform for experimenting with new tableaux algorithms and optimisation techniques.[1] It incorporates most of the standard optimisation techniques, including those introduced in the FaCT system [3], but also employs many novel ones. This includes a new "ToDo list" architecture that is better suited to more complex tableaux algorithms (such as those used to reason with OWL ontologies), and allows for a wider range of heuristic optimisations.

2 Tableaux Reasoning and Architecture

DL systems take as input a knowledge base (equivalently an ontology) consisting of a set of axioms describing constraints on the conceptual schema (often called the Tbox) and a set of axioms describing some particular situation (often called the Abox). They

[1] FaCT++ is avaliable at `http://owl.man.ac.uk/factplusplus`.

are then able to answer both "intensional" queries (e.g., regarding concept satisfiability and subsumption) and "extensional" queries (e.g., retrieving the instances of a given concept) w.r.t. the input knowledge base (KB). For the expressive DLs implemented in modern systems, these reasoning tasks can all be reduced to checking KB satisfiability.

Most modern DL systems are based on tableaux decision procedures, as first introduced by Schmidt-Schauß and Smolka [10], and subsequently extended to deal with ever more expressive logics [1]. Many systems now implement the $\mathcal{SHIQ}$ or $\mathcal{SHOIQ}$ DLs, tableaux algorithms for which were presented in [7, 6]; these logics are very expressive, and correspond closely to the OWL ontology language. In spite of the high worst case complexity of the KB satisfiability problem for these logics (ExpTime-complete and NExpTime-complete respectively), highly optimised implementations have been shown to work well in many realistic (ontology) applications [3].

When reasoning with a KB, FaCT++ proceeds as follows. A first *preprocessing* stage is applied to the KB when it is loaded into reasoner; it is normalised and transformed into an internal representation. During this process several optimisations (that can be viewed as a syntactic re-writings) are applied.

The reasoner then performs *classification*, i.e., computes and caches the subsumption partial ordering (taxonomy) of named concepts. Several optimisations are applied here, mainly involving choosing the order in which concepts are processed so as to reduce the number of subsumption tests performed.

The classifier uses a KB *satisfiability* checker in order to decide subsumption problems for given pairs of concepts. This is the core component of the system, and the most highly optimised one.

3 FaCT++ Optimisations

3.1 Preprocessing Optimisations

Lexical normalisation and *simplification* is a standard rewriting optimisation primarily designed to promote early clash (inconsistency) detection, although it can also simplify concepts and even detect relatively trivial inconsistencies [4]. The basic idea is that all concepts are transformed into a *simplified normal form* (SNF), where the only operators allowed in SNF are negation ($\neg$), conjunction ($\sqcap$), universal restriction ($\forall$) and (qualified) at-most restriction ($\leq$). In FaCT++, the translation into SNF is performed on the fly, during the parsing process. At the same time, some simplifications are applied to concept expressions, including constant elimination (e.g., $C \sqcap \bot \rightarrow \bot$), expression elimination (e.g., $\neg\neg C \rightarrow C$), and subsumer elimination (e.g., $C \sqcap D \rightarrow C$ for D a known subsumer of C).

Absorption is a widely used rewriting optimisation that tries to eliminate General Concept Inclusion axioms (GCIs, axioms in the form $C \sqsubseteq D$, where both C and D are complex concept expressions), as GCIs left in the Tbox invariably lead to a significant decrease in the performance of tableaux based satisfiability/subsumption testing procedures [3]. In FaCT++, GCIs are eliminated by absorbing them into either concept definition axioms (*concept absorption*) or role domain axioms (*role absorption*). Role absorption is particularly beneficial from the point of view of the CD-classification optimisation (see Section 3.3), as it eliminates GCIs without reducing the number of concepts to which CD-classification can be applied.

Told Cycle Elimination is a technique that we assume is used in most modern reasoners, although we know of no reference to it in the literature. Definitional cycles in the Tbox can lead to several problems, and in particular cause problems for algorithms that exploit the told subsumer hierarchy (see Section 3.3). These cycles are, however, often quite easy to eliminate. Assume, for example, that $A_1 \dots A_n$ are named concepts, $C_1 \dots C_n$ are arbitrary concept expressions, and $\bowtie$ is either $\sqsubseteq$ or $\equiv$. The axioms $A_1 \bowtie A_2 \sqcap C_2, A_2 \bowtie A_3 \sqcap C_3, \dots, A_n \bowtie A_1 \sqcap C_1$ include a definitional cycle, because the r.h.s. of the first axiom (indirectly) refers to the name on its l.h.s. The cycle can, however, be eliminated by transforming the axioms into $A_2 \equiv A_1, \dots, A_n \equiv A_1, A_1 \sqsubseteq C_1 \sqcap C_2 \dots \sqcap C_n$.

Synonym Replacement is used to extend simplification possibilities and improve early clash detection. If the only axiom with C on the left hand side is $C \equiv D$, then C is called a *synonym* of D. For a set of concept names, all of which are synonymous, FaCT++ uses a single "canonical" name in all concept expressions in the KB.

FaCT++ first translates all input concepts into SNF, with subsequent transformations being designed to preserve this form. After simplification and absorption, FaCT++ repeatedly performs cycle and synonym elimination steps until there are no further changes to the KB.

3.2 Satisfiability Checking Optimisations

The FaCT++ system was designed with the intention of implementing DLs that include inverse roles, and of investigating new optimisation techniques, including new ordering heuristics. In order to deal more easily with inverse roles, and to allow for more flexible ordering of the tableaux expansion, FaCT++ uses a *ToDo list*, instead of the usual top-down approach, to control the application of the expansion rules [13]. The basic idea behind this approach is that rules may become applicable whenever a concept is added to a node label. When this happens, the relevant node/concept pair is added to the ToDo list. The ToDo list sorts entries according to some order, and gives access to the "first" element in the list. The tableaux algorithm repeatedly removes and processes list entries until either a clash occurs or the list become empty.

Dependency-directed backtracking (Backjumping) is a crucial and widely used optimisation. Each concept in a completion tree label is labelled with a *dependency set* containing information about the branching decisions on which it depends. In case of a clash, the system backtracks to the most recent branching point where an alternative choice might eliminate the cause of the clash.

Boolean constant propagation (BCP) is another widely used optimisation. As well as the standard tableau expansion rules, additional inference rules can be applied to the formulae occurring in a node label, usually with the objective of simplifying them and reducing the number of nondeterministic rule applications. BCP is probably the most commonly used simplification, the basic idea being to apply the inference rule

$$\frac{\neg C_1, \dots, \neg C_n, C_1 \sqcup \dots \sqcup C_n \sqcup C}{C}$$

to concepts in a node labels.

Semantic Branching is another rewriting optimisation, the idea being to rewrite disjunctions of the form $C \sqcup D$ as $C \sqcup (\neg C \sqcap D)$. If choosing C leads to clash, then the

$\neg C$ in the second disjunct (along with BCP) ensures that C will not be added to the node label again by some other nondeterministic expansion.

Ordering Heuristics can be very effective, and have been extensively investigated in FaCT++ [13]. Changing the order in which nondeterministic expansions are explored can result in huge (up to several orders of magnitude) differences in reasoning performance. Heuristics can be used to choose a "good" order in which to try the different possible expansions. In practise, this usually means using heuristics to select the way in which expansion rules are applied to the disjunctive concepts in a node label, with a heuristic function being used to compute the relative "goodness" of each candidate expansion.

Heuristics may select an expansion-ordering based on, e.g., (ascending or descending order of) concept size, maximum quantifier depth, or frequency of usage. In order to reduce the cost of computing the heuristic function, FaCT++ computes and caches relevant values for each concept as the KB is loaded. As no one heuristic performs well in all cases, FaCT++ also selects the heuristics to be used based on an analysis of the structure of the input KB.

3.3 Classification Optimisations

As mentioned above, the focus here is on reducing the number of subsumption tests performed during classification. In FaCT++, this is achieved by both reducing the number of comparisons and by substituting cheaper (but incomplete) comparisons.

Definitional Ordering is a well known technique that uses the syntactic structure of Tbox axioms to optimise the order in which the taxonomy is computed. E.g., given an axiom $C \sqsubseteq D$, with C a concept name, FaCT++ will delay adding C to the taxonomy until all of the concepts occurring in D have been classified. In some cases this technique allows the taxonomy to be computed "top down", thereby avoiding the need to check for subsumees of newly added concepts.

Similarly, the structure of Tbox axioms can be used to avoid (potentially) expensive subsumption tests by computing a set of (trivially obvious) *told subsumers* and *told disjoints* of a concept C. E.g., if the Tbox contains an axiom $C \sqsubseteq D_1 \sqcap D_2$, then FaCT++ treats both D_1 and D_2, as well as all *their* told subsumers, as told subsumers of C, and if the Tbox contains an axiom $C \sqsubseteq \neg D \sqcap \ldots$, then D is treated as a told disjoint of C. The classification algorithm can then exploit obvious (non-) subsumptions between concepts an their told subsumers (disjoints).

Model Merging is a widely used technique that exploits cached partial models in order to perform a relatively cheap but incomplete non-subsumption test. If the cached models for D and $\neg C$ can be merged to give a model of $D \sqcap \neg C$, then the subsumption $C \sqsubseteq D$ clearly does not hold.

Completely Defined Concepts is a novel technique used in FaCT++ to deal more effectively with wide (and shallow) taxonomies [12]. In this case, some concepts in the taxonomy may have very many direct subsumees, rendering classification ordering optimisations ineffective. It is often possible, however, to identify a significant subset of concepts whose subsumption relationships are completely defined by told subsumptions. FaCT++ computes a taxonomy for these concepts without performing any subsumption tests.

Clustering is another technique that addresses the same problem [2]. The idea here is to introduce new "virtual concepts" into the taxonomy in order to produce a deeper and more uniform structure. These concepts are asserted to be equivalent to the union of a number of sibling concepts and are inserted in the taxonomy in between these concepts and their common parent.

4 Discussion and Future Directions

We have presented FaCT++, a reasoner for $\mathcal{SHOIQ}$ (and so OWL DL) which uses a new ToDo list architecture and incorporates a wide range of optimisations, including several novel ones.

Future directions for FaCT++ include both algorithmic and technological improvements. The next version of FaCT++ will support the more expressive $\mathcal{SROIQ}$ DL needed by the OWL 1.1 ontology language (see `http://owl-workshop.man.ac.uk/OWL1_1.html`). Some new optimisations, including optimised reasoning with nominals [11] and more elaborate heuristics are also planned. Regarding technological improvements, we plan to add direct support for OWL's XML syntax, and to parallelise the reasoning process.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. CUP, 2003.
2. V. Haarslev and R. Möller. High performance reasoning with very large knowledge bases: A practical case study. In *Proc. of IJCAI 2001*, pages 161–168, 2001.
3. I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of KR'98*, pages 636–647, 1998.
4. I. Horrocks. Implementation and optimisation techniques. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 306–346. CUP, 2003.
5. I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.
6. I. Horrocks and U. Sattler. A tableaux decision procedure for $\mathcal{SHOIQ}$. In *Proc. of IJCAI 2005*, 2005.
7. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proc. of LPAR'99*, number 1705 in LNAI, pages 161–180, 1999.
8. H. Knublauch, R. Fergerson, N. Noy, and M. Musen. The protégé OWL plugin: An open development environment for semantic web applications. In *Proc. of ISWC 2004*, number 3298 in LNCS, pages 229–243, 2004.
9. A. Rector. Medical informatics. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 415–435. CUP, 2003.
10. M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.
11. E. Sirin, B. C. Grau, and B. Parsia. From wine to water: Optimizing description logic reasoning for nominals. In *Proc. of KR 2006*, 2006. To Appear.
12. D. Tsarkov and I. Horrocks. Optimised classification for taxonomic knowledge bases. In *Proc. of the 2005 Description Logic Workshop (DL 2005)*, 2005.
13. D. Tsarkov and I. Horrocks. Ordering heuristics for description logic reasoning. In *Proc. of IJCAI 2005*, 2005.