Applying an $\mathcal{ALC}$ ABox Consistency Tester to Modal Logic SAT Problems

Volker Haarslev and Ralf Möller

University of Hamburg, Computer Science Department
Vogt-Koelln-Str. 30, 22527 Hamburg, Germany
{haarslev,moeller}@informatik.uni-hamburg.de
http://kogs-www.informatik.uni-hamburg.de/~{haarslev,moeller}

Abstract. In this paper we present the results of applying HAM-ALC, a description logic system for $\mathcal{ALCNR}$, to modal logic SAT problems.

1 Introduction

Research on description logics and modal logics tackles related problems from different viewpoints. One of the recent advances in the development of "fast" description logic systems was the FaCT architecture [4] which focuses on TBox reasoning. Besides optimizations for computing the subsumption hierarchy (e.g. taxonomic encoding and other techniques), the FaCT system is based on optimized algorithms with appropriate data structures for speeding up the basic concept consistency test (dependency-directed backtracking, semantic branching [2] and caching of models). FaCT supports the logic $\mathcal{ALC}$ plus transitive roles, features and role hierarchies. In addition, generalized concept inclusions (GCIs, [1]) as well as cyclic terminologies are handled by preprocessing techniques and specific blocking strategies being used in the concept satisfiability algorithm [4]. Among other improvements for concept satisfiability checking, the importance of extensive model caching for concept terms and appropriate data structures for models is demonstrated by the evaluation results of the DLP system, a reimplementation of the algorithms used in the FaCT architecture [5]. One of the main results of the research on FaCT and DLP is that a well-designed combination of different techniques and strategies is necessary in order to dramatically increase system performance in the average case. However, neither FaCT nor DLP deals with ABoxes.

In the following we discuss the optimized description logic system HAM-ALC, which has been developed to extend the facilities offered by FaCT. In addition, HAM-ALC supports ABox reasoning for the language $\mathcal{ALCNR}$ which is presented in [1] ($\mathcal{ALC}$ with number restrictions, role conjunction or role hierarchies as well as GCIs but without transitive roles). Besides optimizations for TBoxes, it provides optimized implementations for the well-known inference problems ABox consistency checking, instance checking, realization, instance retrieval [1].

In the following we briefly sketch how known optimization techniques for concept consistency checking can be exploited for building efficient *ABox consistency checking* architectures. Afterwards, we demonstrate that this architecture can also be used for effectively solving average-case modal logic SAT problems. These first tests indicate that the implementation overhead inherent in a system for expressive description logics supporting ABoxes can be reduced to a minimum compared to a concept consistency checking architecture provided by, for instance, DLP.

2 Basic Architecture of HAM-ALC

Similar to the techniques used in FaCT a preprocessing phase transforms concept expressions into negation normal form, removes duplicates, performs obvious simplifications, detects obvious clashes, flattens nested and/or expressions, normalizes the order of disjuncts and conjuncts, and provides a unique identification for all concepts which are structurally equal. For each concept, its negated counterpart is precomputed in order to support a fast access to negations of concepts (required for clash detection, see below).

ABox constraints consist of individual assertions ($i : \mathsf{C}$) as well as role assertions ($\langle i_1, i_2 \rangle : \mathsf{R}$). The ABox consistency checker has to deal with (possibly cyclic) graph structures at least in a finite part of the ABox. Thus, HAM-ALC has to explicitly represent role assertions as well as individuals. For an individual assertion HAM-ALC represents its name and non-negated preprocessed concept expression with a separated negation sign and a set of dependency ABox constraints documenting the origin of this assertion. The dependency constraints are required for dependency-directed backtracking (see below). In order to facilitate extensibility HAM-ALC does not use special "encoding tricks" for representing concepts but uses record structures to store relevant information. It normalizes or-, all-, and number restriction concepts of constraints into their equivalent negated form (e.g. $(\mathsf{C}_1 \sqcup \mathsf{C}_2) \rightarrow \neg(\neg\mathsf{C}_1 \sqcap \neg\mathsf{C}_2)$, $(\forall\ \mathsf{R}\ \mathsf{C}) \rightarrow \neg(\exists\ \mathsf{R}\ \neg\mathsf{C})$, and $(\exists_{\leq n}\ \mathsf{R}) \rightarrow \neg(\exists_{\geq n+1}\ \mathsf{R})$) while representing the negation sign of the concept as part of the constraint itself. This architectural decision helps to speed up the usual clash checks that test whether two constraints $i : \mathsf{C}_1$ and $i : \mathsf{C}_2$ exist such that $\mathsf{C}_1 \sqcap \mathsf{C}_2$ is equal to $\bot$).

Constraints are considered as deterministic if their concept term is either atomic, an and-concept, or an or-concept with exactly one open disjunct (with an unknown truth value). The optimized algorithm treats the consistency test of ABox constraints generated by some- and at-least constraints as isolated subproblems if value and at-most restrictions are carefully handled (see the calculus presented in [1]).

2.1 Optimization Techniques

Or-constraints are a major source of complexity in tableaux expansion. Two major optimization strategies are embedded into the architecture of HAM-ALC that deal with this complexity. The first technique is called *semantic branching*, the second one is *dependency-directed backtracking*. A third strategy tries to avoid the recomputation of identical or similar subtableau by using a so-called "model caching and merging technique" that possibly replaces the tableau satisfiablity test by operating on cached models for concepts. We briefly review these techniques and explain their integration into HAM-ALC.

Semantic Branching In contrast to syntactic branching, where redundant search spaces may be repeatedly explored, semantic branching uses a *splitting rule* which divides the original problem into two smaller disjoint subproblems (see [2] for a discussion). Semantic branching is usually supported by various techniques intended to speed up the search.

A *lookahead algorithm* or *constraint propagator* is applied to reduce the order of magnitude of the open search space. Thus, after every tableau expansion step HAM-ALC propagates the truth value of the newly added constraint into all open disjuncts of all or-constraints with an unknown truth value. As a result of this step, or-constraints might be recognized as satisfied (i.e. one disjunct is satisfied), deterministic (i.e. exactly one disjunct remains open), or might even clash (all disjuncts are unsatisfied).

Various *heuristics* are used to select the next or-constraint and one of its disjuncts for processing. HAM-ALC employs a dynamic selection scheme. The oldest-first nesting strategy is used for selecting one or-constraint with at least two open disjuncts. The selection of a disjunct from this or-constraint is achieved by counting the negated and non-negated occurrences for each open disjunct in all other open or-constraints. These numbers are used as input for a priority function that selects the disjunct. The priority function is adopted from FaCT and achieves the following goals. It prefers disjuncts that occur frequently in unexpanded binary constraints and balanced or-constraints (i.e. containing a similar number of negated and non-negated occurrences of the same disjunct) but discriminates between unbalanced or-constraints. In order to perform individual-specific counting very quickly, HAM-ALC precomputes data structures for cross-referencing open or-constraints that contain this concept in negated and/or non-negated form. Once a disjunct is selected, the priority function is also used to determine which branch of the search tree is tried first by the splitting rule.

Dependency-directed Backtracking Naive backtracking algorithms often explore regions of the search space rediscovering the same contradictions repeatedly. An integral part of the HAM-ALC architecture is a dependency management system. It records the dependencies of every constraint, i.e. whenever a constraint is created, its precondition constraints are saved as a dependency set. This set is employed by the dependency-directed backtracking technique of HAM-ALC in order to reduce the search space.

Whenever a clash occurs, the union of the dependency sets of the clash culprits (referred to as clash dependency set) is recorded and backtracking is started.

When a semantic branching point is encountered during backtracking, HAM-ALC checks whether this or-constraint is responsible for a clash culprit (i.e. is a member of the clash dependency set). If the or-constraint is not found, this branching point can be safely bypassed. In case the or-constraint is found, either the remaining semantic alternative is tried or this disjunct is considered as unsatisfiable in the current subtree. The backtracking continues but removes the current or-constraint from the clash dependency set and adds the saved clash dependency set of the first clashed alternative. This technique was first realized in the FaCT [4] system and is extended in the HAM-ALC architecture for dealing with different individuals.

Model-based Satisfiability Tests The third major strategy tries to avoid the recomputation of identical or similar subtableaux (caused by a some- or an at-least constraint and the corresponding all- and at-most constraints) by using operations on cached "models" for concepts. The test whether a concept C subsumes a concept D is preceded or may be even replaced by a merging test for the models of $\neg\mathsf{C}$ and D. This technique was first developed for the FaCT system for $\mathcal{ALC}$. HAM-ALC extends this technique for $\mathcal{ALCNR}$ and refines it in several ways: (1) In contrast to FaCT it deals with *deep* models and introduces and exploits *deterministic* models. (2) Every satisfiability test of a subtableau is preceded by a model merging test working with either deep or flat models. (3) The concept subsumption test is devised as a two-level procedure first trying a novel structural subsumption test (see below) that is optionally followed by a regular tableaux satisfiability test.

A model of a concept is computed by applying the standard satisfiability test. In case of a failure this incoherent concept is associated with the $\perp$-model. Otherwise HAM-ALC constructs and caches a model from the final tableau. A *model* consists of a *concept set* containing every (negated) atomic, some-, at-least, all-, and at-most concept occurring in the final constraint set of the tableau. And- and or-concepts may be safely ignored due to their decomposition by the tableaux rules. A model is marked as *deterministic* if the constraint set contains no or-constraint and no at-most constraint that caused fork elimination.

The standard *flat model merging test* (due to FaCT) for a set of models $\mathcal{M} = \{M_1, \ldots, M_n\}$ works as follows. The concept sets of every pair $\langle M_i, M_j \rangle$ (with $M_i, M_j \in \mathcal{M}$ and $i \neq j$, $1 \leq i, j \leq n$) are mutually checked for a potential clash. If either a pair $\langle C_i, C_j \rangle$ (with $C_i \in M_i$, $C_j \in M_j$) of clashing atomic concepts or of potentially interacting some- and all-concepts via a common role R is found, the test returns *unmergable*. Otherwise it returns *mergable*. The flat model merging test is sound but *not* complete. Thus, it precedes every concept subsumption and subtableau satisfiability test and replaces it if the answer is *mergable*. HAM-ALC extends this technique for $\mathcal{ALCNR}$ in two ways. Its model merging test correctly deals with number restriction concepts and keeps track of deterministic models and becomes sound *and* complete if only deterministic models are involved. It realizes a *deep* model merging test that recursively checks the models of potentially interacting some- and all-concepts. HAM-ALC tries to maximize the use of deterministic models for concept subsumption tests.

Table 1. TANCS'99 selected reference problems of the modal pspace division.

Problem	Clauses	Variables	Depth	Runtime (10ms)
bounded CNF	8	4	2	0
	16	4	2	2
	32	4	2	6
bounded CNF modK	8	4	2	2
	16	4	2	5
	32	4	2	14
unbounded QBF	8	2	3	27
	16	2	3	33
	32	2	3	53
unbounded QBF modK	8	2	3	24
	16	2	3	36
	32	2	3	61

3 Implementation Language and Special Features

HAM-ALC is implemented in Common Lisp and has been tested with Macintosh Common Lisp, Allegro Common Lisp (SunOS, Windows, Linux). HAM-ALC provides a Web-based interface [3].

4 First Results on a Performance Analysis

For TANCS-99 we have tested HAM-ALC on the modal PSPACE division reference problems (see Table 1). The runtimes (in 10ms) are computed based on the files provided by TANCS-99. For each parameter setting (see Table 1) the computation results of the 16 problem instances are averaged (geometric mean). We have run the system on a Sun Ultra Sparc 2 (300 MHz) with Allegro CL 5.0.

References

1. M. Buchheit, F.M. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1:109–138, 1993.
2. J.W. Freeman. *Improvements to propositional satisfiability search algorithms.* PhD thesis, University of Pennsylvania, Computer and Information Science, 1995.
3. V. Haarslev, R. Möller, and A.-Y. Turhan. Implementing an $\mathcal{ALCRP}(\mathcal{D})$ ABox reasoner: Progress report. In E. Franconi et al., editor, *Proceedings of the International Workshop on Description Logics (DL'98), June 6-8, 1998, Trento, Italy*, pages 82–86, June 1998.
4. I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics.* PhD thesis, University of Manchester, 1997.
5. I. Horrocks and P.F. Patel-Schneider. FaCT and DLP: Automated reasoning with analytic tableaux and related methods. In *Proceedings International Conference Tableaux'98*, pages 27–30, 1998.