Consistency Testing: The RACE Experience

Volker Haarslev and Ralf Möller

University of Hamburg, Computer Science Department
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany

Abstract. This paper presents the results of applying RACE, a description logic system for $\mathcal{ALCNH}_{R^+}$, to modal logic SAT problems. Some aspects of the RACE architecture are discussed in detail: (i) techniques involving caching and (ii) techniques for dealing with individuals.

1 Introduction to the RACE Architecture

The description logic (DL) $\mathcal{ALCNH}_{R^+}$ [5] extends the logic $\mathcal{ALCHf}_{R^+}$ (see [8] for a concept consistency calculus) by adding number restrictions. The inference services supported by RACE for TBoxes and ABoxes are described in [6]. In this paper, due to space restrictions, we assume that the reader is familiar with DLs.

The ABox consistency algorithm implemented in the RACE system is described as a tableaux calculus in [5]. However, optimized search techniques are required in order to guarantee good average-case performance. The RACE architecture incorporates the following standard optimization techniques: dependency-directed backtracking [12] and DPLL-style semantic branching (see [3] for recent results and for an overview of the literature). Among a set of new optimization techniques, the integration of these techniques into DL reasoners for concept consistency has been described in [7] (see also [2] for a discussion of ideas to integrate intelligent backtracking into terminological logics). The implementation of these techniques in the ABox reasoner RACE differs from the implementation of other DL systems (e.g. FaCT or DLP [10]), which provide only concept consistency (and TBox) reasoning. The latter systems have to consider only so-called "labels" (sets of concepts) whereas an ABox prover such as RACE has to explicitly deal with individuals (nominals).

The techniques for TBox reasoning described in [1] (marking and propagation as well as lazy unfolding) are also supported by RACE. As indicated in [4], the architecture of RACE is inspired by recent results on optimization techniques for TBox reasoning [9], namely transformations of axioms (generalized concept inclusions, GCIs), model caching and model merging.

RACE is implemented in Common Lisp and can be copied for research purposes: http://kogs-www.informatik.uni-hamburg.de/~moeller/race/race.html.

2 Results on the TANCS Comparison Problems

The TANCS QBF benchmarks are transformed into concept consistency tests. The runtimes of RACE on the comparison problems with different encodings are documented in Table 1. All tests have been run on a Macintosh Powerbook G3

Table 1. Runtimes for the TANCS'2000 comparison problems (runtimes include reading times): S = definitely satisfiable, T = timed out. The percentage of definitely unsatisfiable problems is U = 100 – S – T. The number of instances provided for the TANCS comparison problems of different type is indicated in parentheses.

K	C	V	D	Time (10ms)	S (%)	T (%)
				QBF cnf (4)		
4	10	4	4	2481	100	0
4	10	4	6	10455	0	100
4	20	4	4	4246	75	0
4	20	4	6	10577	0	100
4	30	4	4	776	0	0
4	30	8	4	11443	0	100
				QBF cnf modS4 (64)		
4	20	2	2	2277	14	6
				QBF cnf SSS (8)		
4	10	4	4	23	100	0
4	10	4	6	31	100	0
4	10	8	4	46	100	0
4	10	8	6	82	100	0
4	10	16	4	99	100	0
4	10	16	6	120	100	0
4	20	4	4	48	75	0
4	20	4	6	68	100	0
4	20	8	4	104	100	0
4	20	8	6	296	100	0
4	20	16	4	297	100	0
4	20	16	6	735	100	0
4	30	4	4	74	25	0
4	30	4	6	116	88	0
4	30	8	4	246	100	0
4	30	8	6	733	100	0
4	30	16	4	1647	100	0
4	30	16	6	5206	25	75
4	40	4	4	61	12	0
4	40	4	6	112	25	0
4	40	8	4	414	25	0
4	40	8	6	3344	88	0
4	40	16	4	3661	75	25
4	40	16	6	8932	25	75
4	50	4	4	71	0	0
4	50	4	6	128	0	0
4	50	8	4	716	0	0
4	50	8	6	3306	62	12
4	50	16	4	9075	25	75
4	50	16	6	9896	12	88

K	C	V	D	Time (10ms)	S (%)	T (%)
				QBF cnf Ladn (8)		
4	10	4	4	425	100	0
4	10	4	6	4487	88	12
4	10	8	4	9644	25	75
4	10	8	6	10191	0	100
4	20	4	4	940	100	0
4	20	4	6	9360	12	88
4	20	8	4	10205	0	100
4	20	8	6	10306	0	100
4	30	4	4	857	25	0
4	30	4	6	9710	0	88
4	30	8	4	10276	0	100
4	30	8	6	10433	0	100
4	40	4	4	508	0	0
4	40	4	6	3680	0	38
4	40	8	4	8919	0	88
4	40	8	6	10557	0	100
4	50	4	4	616	12	0
4	50	4	6	1851	0	12
4	50	8	4	10421	0	100
4	50	8	6	10689	0	100
				PSAT cnf (8)		
4	20	4	1	11	100	0
4	20	4	2	29	100	0
4	20	8	1	10	100	0
4	20	8	2	78	100	0
4	30	4	1	14	100	0
4	30	4	2	132	100	0
4	30	8	1	16	100	0
4	30	8	2	2987	62	38
4	40	4	1	23	100	0
4	40	4	2	364	100	0
4	40	8	1	26	100	0
4	40	8	2	10651	12	88
4	50	4	1	30	88	0
4	50	4	2	2088	62	25
4	50	8	1	32	100	0
4	50	8	2	10017	0	100

with 333 MHz. A timeout is set to 100secs. In case of a timeout, the runtime (100secs) is included into the geometric mean. For some of the TANCS'2000 comparison problem types, the set of benchmarks contains harder problems, which are not reported in Table 1. For these problems all instances are timed out, hence the runtime for each problem is 100secs.

The PSAT problems of the TANCS comparison problems are defined with a set of axioms. Axioms are represented as generalized concept inclusions (GCIs). In the RACE system, well-known as well as novel transformations on GCIs are employed in order to reduce runtimes. The goal of the transformations (see [7]) is to reduce the number of GCIs by transforming GCIs in order to derive concept definitions, i.e. a pair of GCIs $\mathsf{A} \sqsubseteq \mathsf{C}$ and $\mathsf{C} \sqsubseteq \mathsf{A}$ or primitive concept definitions $\mathsf{A} \sqsubseteq \mathsf{C}$ (with no $\mathsf{C} \sqsubseteq \mathsf{A}$) such that A is not mentioned on the left-hand side of another GCI in the TBox. Furthermore, as a novel transformation technique, in RACE axioms for declaring the disjointness of atomic concepts and axioms for domain and range restrictions for roles are absorbed. These constructs are treated in a special way by the RACE architecture.

In the PSAT problems the axioms are of the form $\top \sqsubseteq \mathsf{C}$ where C is a complex concept expression (i.e. a complex modal logic formula). Unfortunately, in the

case of PSAT the number of axioms (GCIs) can only be reduced by detecting common subexpressions. The derivation of (primitive) concept definitions is not possible due to the structure of the formulae (see Table 1 for the runtime results).

3 An Important Optimization Technique: Caching

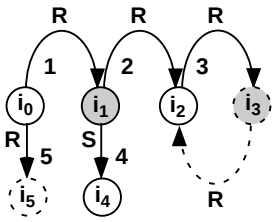
Caching of intermediate computation results is a necessary prerequisite to prove the (in)consistency of many concepts terms [9]. In particular, solutions for some of the TANCS problems mentioned above cannot be computed within the time limit without caching. For instance, the runtimes for the PSAT problems with depth 2 (see Table 1) increase by one order of magnitude.

Caching is not a trivial optimization technique. Solving a consistency problem $\{\mathsf{i}_0 : \mathsf{E}\}$ w.r.t. the following (cyclic) inclusion axioms demonstrates that caching must depend on the context:

$$\mathsf{C} \sqsubseteq (\exists\,\mathsf{R}\,.\,\mathsf{D}) \sqcap (\exists\,\mathsf{S}\,.\,\mathsf{X}) \sqcap \forall\,\mathsf{S}\,.\,(\neg\mathsf{X} \sqcap \mathsf{A}),\ \ \mathsf{D} \sqsubseteq \exists\,\mathsf{R}\,.\,\mathsf{C},\ \ \mathsf{E} \sqsubseteq (\exists\,\mathsf{R}\,.\,\mathsf{C}) \sqcup (\exists\,\mathsf{R}\,.\,\mathsf{D})$$

The proof steps are presented in Figure 1. In the figure the sequence of "expansion" steps is indicated with numbers. In step 1, the initial problem $\{\mathsf{i}_0 : \mathsf{E}\}$ is presented. Since there is an axiom for E involving a disjunction we get two constraint systems (see step 2). Let us assume the first alternative is tried first. This leads to a subconstraint system (step 3). The assertion from step 3 is expanded w.r.t. the axioms and we get the constraint system in step 4. The first some-constraint is expanded first. In step 5 the corresponding subconstraint system is considered. The right-hand side of the axiom for D is inserted (step 6). The some-constraint yields another subconstraint system (step 7). Due to the blocking strategy [4], the constraint system in step 7 is not expanded (see the constraint system in step 3 with the "blocking witness" i_1). In Figure 1 the canonical interpretation is indicated with a dashed arrow.

An often-employed strategy is to cache intermediate results, i.e. the satisfiability of D is stored as a so-called pseudo model $\{\mathsf{D}, \exists\,\mathsf{R}\,.\,\mathsf{C}\}$. Let us assume at the end of step 7 a pseudo model for D is stored as indicated above. In our example, there are some proof steps pending. In step 8 the remaining constraints from step 4 are considered. Obviously, the second some-constraint for the role S together with the value restriction for S causes a clash. Therefore, the second



1. $\{\mathsf{i}_0 : \mathsf{E}\}$
2. $\{\mathsf{i}_0 : \mathsf{E}, \mathsf{i}_0 : \exists\,\mathsf{R}\,.\,\mathsf{C}\} \vee \{\mathsf{i}_0 : \mathsf{E}, \mathsf{i}_0 : \exists\,\mathsf{R}\,.\,\mathsf{D}\}$
3. $\{\mathsf{i}_1 : \mathsf{C}\}$
4. $\{\mathsf{i}_1 : \mathsf{C}, \mathsf{i}_1 : \exists\,\mathsf{R}\,.\,\mathsf{D}, \mathsf{i}_1 : \exists\,\mathsf{S}\,.\,\mathsf{X}, \mathsf{i}_1 : \forall\,\mathsf{S}\,.\,\neg\mathsf{X} \sqcap \mathsf{A}\}$
5. $\{\mathsf{i}_2 : \mathsf{D}\}$
6. $\{\mathsf{i}_2 : \mathsf{D}, \mathsf{i}_2 : \exists\,\mathsf{R}\,.\,\mathsf{C}\}$
7. $\{\mathsf{i}_3 : \mathsf{C}\}$
8. $\{\mathsf{i}_4 : \mathsf{X}, \mathsf{i}_4 : \mathsf{A}, \mathsf{i}_4 : \neg\mathsf{X}\}$
9. $\{\mathsf{i}_5 : \mathsf{D}\}$

Fig. 1. Caching example with blocking (see text).

alternative in step 2 has to be considered. The corresponding subconstraint system is presented in step 9. If the consistency of D is checked by examining a cache entry for D, the overall result will be "E is consistent". Obviously, this is erroneous. The reason is that the caching principle described above does not consider the dependency on the satisfiability of C. Therefore, a dependency tracking mechanism for cache entries is implemented in RACE. Once the system detects the inconsistency of a concept (or constraint system) on which a cached pseudo model is dependent, the corresponding cache entries are (recursively) removed.

Caching of a pseudo model for a specific concept (model caching) is a strategy which is implemented in the FaCT system and the DLP system [9]. To the best of our knowledge, only the DLP system offers an additional caching technique. If, during a certain tableaux proof for the consistency of a concept term, a some-constraint (e.g. $\exists\,\mathsf{R}\,.\,\mathsf{C}$ interacts with all-constraints $\forall\,\mathsf{R}\,.\,\mathsf{C}$ and $\forall\,\mathsf{R}\,.\,\mathsf{D}$), then another strategy is to cache the result of checking the consisteny of the subproblem $\{\mathsf{i}_{\mathsf{new}}:\mathsf{C}, \mathsf{i}_{\mathsf{new}}:\mathsf{D}, \mathsf{i}_{\mathsf{new}}:\mathsf{E}\}$. Again, the dependency tracking mechanism implemented in RACE ensures correct behavior of the so-called "subtableaux caching" strategy in the case of blocking. To the best of our knowledge, details about the techniques employed in DLP have not been published yet.

If there exists no cache entry for a subproblem such as $\{\mathsf{i}_{\mathsf{new}}:\mathsf{C}, \mathsf{i}_{\mathsf{new}}:\mathsf{D}, \mathsf{i}_{\mathsf{new}}:\mathsf{E}\}$, then it might be possible check out whether the models of C, D and E can be merged (see [7] for an introduction to model merging). However, for the TANCS benchmarks this technique was disabled because the overhead involved in this test caused the runtimes to increase.

4 Optimizations for ABox Reasoning

Unfortunately, the TANCS benchmarks only consider concept consistency problems (possibly w.r.t. axioms). The RACE architecture has been developed also for ABox reasoning with individuals. Thus, there is some overhead involved if only concept consistency tests are computed with RACE. As the runtime results indicate, the overhead costs of pure consistency reasoning with an ABox reasoner are minimal. However, ABox reasoning itself requires additional optimization techniques in order to ensure adequate average-case performance. For instance, for computing the direct types (most-specific atomic concepts) of an individual a during ABox realization, multiple consistency problems for ABoxes $\mathcal{A} \cup \{\mathsf{a}:\neg\mathsf{A}_\mathsf{i}\}$ with different atomic concepts A_i have to be solved. In order to optimize ABox realization (see [4] for other ABox optimization techniques) we use a transformation technique for tree-like ABoxes, called *contraction* w.r.t. an individual a. The idea is to maximize the effect of caching. To speed up the tests we transform $\mathcal{A}$ in such a way that acyclic, tree-like "role paths" between individuals are represented by an appropriate existential restriction. The corresponding concept and role assertions "representing" the role paths are deleted from the ABox. The following contraction rule is applied to $\mathcal{A}$ as often as possible. The final ABox is called $\mathcal{A}'$. Then, the consistency of $\mathcal{A}' \cup \{\mathsf{a}:\neg\mathsf{A}_\mathsf{i}\}$ is checked. Obviously, $\mathcal{A}$ (without $\mathsf{a}:\neg\mathsf{A}_\mathsf{i}$ being added) must be tested for consistency without contraction beforehand.

RC Contraction Rule for $\mathcal{ALCNH}_{R^+}$.
Premise: $\exists(\mathsf{i},\mathsf{j}):\mathsf{R} \in \mathcal{A}, \mathsf{j}:\mathsf{C}_1 \in \mathcal{A}, \ldots, \mathsf{j}:\mathsf{C}_\mathsf{n} \in \mathcal{A}$:
$[\mathsf{j} \neq \mathsf{a},\ (\neg\exists\,(\mathsf{j},\mathsf{k}):\mathsf{R}' \in \mathcal{A}),$
$(\neg\exists\,(\mathsf{i},\mathsf{j}):\mathsf{R}'' \in \mathcal{A} : \mathsf{R}'' \neq \mathsf{R}), (\neg\exists\,(\mathsf{l},\mathsf{j}):\mathsf{R}''' \in \mathcal{A} : \mathsf{l} \neq \mathsf{i})$
$(\neg\exists\,(\mathsf{i},\mathsf{o}):\mathsf{S} \in \mathcal{A} : \mathsf{o} \neq \mathsf{j}, \mathsf{R}^\uparrow \cap \mathsf{S}^\uparrow \neq \emptyset),$
$(\neg\exists\,\mathsf{j}:\mathsf{C}_{\mathsf{n}+1} \in \mathcal{A} : \forall\,\mathsf{i} \in 1..\mathsf{n} : \mathsf{C}_{\mathsf{n}+1} \neq \mathsf{C}_\mathsf{i})\,]$
Consequence: $\mathcal{A} := (\mathcal{A} \setminus \{(\mathsf{i},\mathsf{j}):\mathsf{R}, \mathsf{j}:\mathsf{C}_1, \ldots, \mathsf{j}:\mathsf{C}_\mathsf{n}\}) \cup \{\mathsf{i}:\exists\,\mathsf{R}\,.\,\mathsf{C}_1 \sqcap \ldots \sqcap \mathsf{C}_\mathsf{n}\}$

Empirical tests with automatically generated ABox problems indicate that the role path contraction technique is very effective for tree-like ABoxes. The contraction technique can give a speed gain of about one order of magnitude (see [4] for a set of benchmarks and evaluation results).

References

1. F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.J. Profitlich. An empirical analysis of optimization techniques for terminological representation systems. *Applied Intelligence*, 2(4):109–138, 1994.
2. D. Drollinger. Intelligentes Backtracking in Inferenzsystemen am Beispiel Terminologischer Logiken (in German). Document D-93-21, Deutsches Forschungszentrum für Künstliche Intelligence, Germany, 1993.
3. J.W. Freeman. *Improvements to propositional satisfiability search algorithms*. PhD thesis, University of Pennsylvania, Computer and Information Science, 1995.
4. V. Haarslev and R. Möller. An empirical evaluation of optimization strategies for ABox reasoning in expressive description logics. In Lambrix et al. [11], pages 115–119.
5. V. Haarslev and R. Möller. Expressive abox reasoning with number restrictions, role hierachies, and transitively closed roles. In A.G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proc. of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR'2000)*, 2000.
6. V. Haarslev, R. Möller, and A.-Y. Turhan. RACE User's guide and reference manual version 1.1. Technical Report FBI-HH-M-289/99, University of Hamburg, Computer Science Department, October 1999.
7. I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.
8. I. Horrocks. Using an expressive description logic: FaCT or fiction? In A.G. Cohn, L. Schubert, and S. Shapiro, editors, *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, June 2-5, 1998*, pages 636–647, June 1998.
9. I. Horrocks and P. Patel-Schneider. Optimising description logic subsumption. *Journal of Logic and Computation*, 9(3):267–293, June 1999.
10. 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.
11. P. Lambrix et al., editor. *Proceedings of the International Workshop on Description Logics (DL'99), July 30 - August 1, 1999, Linköping, Sweden*, June 1999.
12. R.M. Stallman and G.J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9(2):135–196, 1977.