An Empirical Evaluation of Optimization Strategies for ABox Reasoning in Expressive Description Logics

Volker Haarslev and **Ralf Möller**
University of Hamburg, Computer Science Department
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
http://kogs-www.informatik.uni-hamburg.de/~<name>/

Abstract

This paper presents an evaluation of a new description logic reasoner called RACE which implements TBox and ABox reasoning for the description logic $\mathcal{ALCNH}_{R^+}$ supporting number restrictions, role hierarchies, and transitively closed roles. Tests on benchmark ABoxes indicate a speedup of several orders of magnitude compared to previous systems.

1 Introduction

This paper investigates techniques for optimizing instance checking and realization in the new DL architecture called RACE (Reasoner for ABoxes and Concept Expressions). RACE supports TBox and ABox reasoning for the description logic $\mathcal{ALCNH}_{R^+}$, i.e. $\mathcal{ALC}$ with number restrictions, role hierarchies and transitive roles (syntactic restrictions: (i) no number restrictions are possible for transitive roles, (ii) for any role which has a transitive subrole no number restrictions are supported). This DL also provides general concept inclusions. Furthermore, in RACE the unique name assumption holds for ABox individuals. The implementation of this logic is based on a sound and complete ABox tableaux calculus given in [3]. RACE is a successor of HAM-ALC [2]. The instance checking and realization algorithms of RACE are based on an optimized *ABox* consistency tester whose architecture is inspired by the results of the work on FaCT and DLP [4]. Note that the description logic languages supported by these systems are different.

This paper discusses the application of algorithms for transforming ABoxes such that the optimization techniques of the RACE ABox consistency reasoner can be exploited to speed up instance checking and realization. The performance gain of different techniques is indicated. With RACE we demonstrate an advance in the state of the art of hybrid reasoning with TBoxes and ABoxes in order to make description logics usable for solving practical problems that could not be tackled before.

2 ABox Consistency Checking

ABox assertions are either concept assertions $i : C$ or role assertions $(i, j) : r$. These assertions are also called ABox constraints. The ABox consistency checker has to deal with (possibly cyclic) graph structures at least in a finite part of the ABox. Thus, RACE has to explicitly represent concept and role assertions as well as individuals. For a concept assertion RACE represents its name and non-negated preprocessed concept expression with a separated negation sign and a set of ABox constraints documenting the "origin" of this assertion. The "origin" constraints are called dependencies. For details on the computation of the set of dependencies from the preconditions of the tableaux rules see [4]. The dependency constraints are required for implementing algorithms for informed search.

2.1 Techniques for Informed Search

Or-constraints and number restrictions (especially in combination with hierarchical roles) are a major source of complexity in tableaux expansion. Two major state-of-the-art optimization strategies are embedded into the architecture of RACE 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 $\exists$-constraints using a so-called "model caching and merging technique" that possibly replaces the tableaux satisfiablity test by operating on cached tableaux data structures for concept constraints. We briefly review these techniques and explain their integration into RACE.

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. Semantic branching is supported by various techniques intended to speed up the search. RACE employs similar techniques as described in [4] but these techniques have been adapted to ABox reasoning. Index structures are set up for dealing with different individuals.

Dependency-directed Backtracking

Naive backtracking algorithms often explore regions of the search space rediscovering the same contradictions repeatedly. An integral part of the RACE 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 RACE in order to reduce the search space. This technique was first realized in the FaCT system and is extended in the RACE architecture for dealing with number restrictions and ABox individuals. Number restrictions introduce additional choice points and are treated by non-deterministic tableaux rules [3]. In a search procedure, corresponding choice points for appropriately "unifying" individuals have to be set up.

Model-based Satisfiability Tests

The third major optimization 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. 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 RACE 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 its constraint set contains no or-constraint and no at-most constraint. RACE realizes a so-called *deep* model merging test that recursively checks the models for potentially interacting some- and all-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 $\neg C$ and D. If the models are mergable, C does not subsume D. This technique was first realized in the FaCT system for $\mathcal{ALCH}f_{R^+}$. RACE extends this technique for $\mathcal{ALCNH}_{R^+}$. The deep model merging test is realized as an incomplete structural consistency test for the language $\mathcal{ALENH}$ ($\mathcal{ALE}$ plus number restrictions and role hierarchies). RACE's 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. If non-$\mathcal{ALENH}$ language constructs are encountered, the structural subsumption test returns "not mergable." In general, if non-deterministic models are involved, a negative answer is equivalent to "do not know." Thus, in this case the full sound and complete $\mathcal{ALCNH}_{R^+}$ tableaux calculus is used.

2.2 ABox consistency vs. concept consistency?

It might be tempting to argue for the development of concept consistency testing algorithms and transforming ABox consistency to concept consistency via so-called precompletion techniques. Let us consider the example ABox $A = \{(i,k) : r,\ (j,k) : r,\ i : (\forall r.A) \sqcup (\forall r.B), j : C \sqcup D, k : \neg A, k : \neg B\}$ where C and D might be complex concept terms. An effective search procedure has to use backtracking to exhaustively explore all possible alternatives. For instance, the case $i : \forall r.A$ must be explored. Another choice point is the disjunction $j : C \sqcup D$. Let us assume the system tries $j : C$ before considering constraints for k (note that C might contain value constraints). Afterwards, the concept constraint $i : \forall r.A$ is treated in combination with the role constraint $(i,k) : r$. After some additional expansion steps, this part of the search tree will lead to a clash because $k : A$ and $k : \neg A$ will be elements of the ABox. Now, if the system backtracks to the choice point $j : C \sqcup D$ and tries $j : D$ it is bound to detect the same clash for k again. Since D can be a very complex concept term, many expansion steps are definitely waisted. This situation is avoided with dependency-directed backtracking, i.e. the system detects that $j : C$ is not involved in the clash and backtracking is set up again for trying $i : \forall r.B$. Note that this backtracking point is found even if the expansion of C causes additional constraints to be propagated from j to k.

In a transformation approach with precompletion it is not guaranteed that the resulting concept term directly suggests this kind of possible search space cutoff. At least, it is very likely that additional transformation steps are necessary, which introduce additional overhead. Thus, due to our experiences, for practical purposes, it is better to use a "native" ABox consistency tester whose dependency structures might also be used for explanation purposes.

3 TBox Classification, Instance Checking and Realization

Besides the ABox consistency problem, the TBox classification, instance checking and realization problems are important inference tasks. The ABox realization process finds the most specific concepts of an individual by traversing the TBox subsumption hierarchy. Basically, starting from the top node this procedure repeatedly calls the instance checking procedure in order to test whether the individual in question is an instance of a certain concept represented by a node in the subsumption hierarchy. Thus, instance checking and, therefore, ABox consistency should be "fast." We test the performance of the RACE system with a set of benchmark problems based on problem defined in the DL'98 system comparison [5].

Table 1. Result of performing the ABox tests with RACE in different optimization modes (see text, times are in seconds).

Test	max	icdp 1	icdp		rcdp		rdp		rcd	
k_branch_n	4	0.33	4	57	4	355	2	33	4	355
k_d4_n	4	0.12	4	114	4	514	2	35	4	537
k_dum_n	21	0.26	21	55	19	508	11	292	19	508
k_grz_n	10	0.07	10	4	9	210	9	393	9	209
k_lin_n	10	0.02	10	2	9	497	9	488	9	497
k_path_n	4	2.97	4	218	3	423	1	13	3	424
k_ph_n	7	0.03	6	6	6	135	6	214	6	135
k_poly_n	8	1.82	8	151	5	416	2	76	5	403
k_t4p_n	5	3.96	5	450	3	228	1	32	3	211

3.1 The Benchmark Problems

The initial DL'98 ABox benchmarks are given as a set of inference problems consisting of a TBox, an ABox and a set of instance checking queries. However, for optimized systems such as RACE, these DL'98 ABox benchmarks are far too simple, i.e. the execution time is almost not measurable. Even for the hardest problems the runtimes are below 3 seconds (realization inclusive). Therefore, we have compiled a new set of benchmarks for $\mathcal{ALC}$ ABoxes based on more complex TBoxes as before. Each benchmark now consists of a set of subproblems which, in turn, consist of an $\mathcal{ALC}$ TBox, an $\mathcal{ALC}$ ABox and a set of instance checking problems which should evaluate to true.

The new benchmark generator is based on the DL'98 TBox benchmarks [5]. From these benchmarks, we generated a set of nine new ABox benchmark problem sets (see Table 1, from now on we refer to these example problems as "the benchmarks"). The names of the benchmarks are listed in the first column. The problems are generated in the following way: The test concept of the corresponding TBox is checked for consistency. Since it is consistent, there exists a model. From these models, a set of ABox assertions is computed in such a way that the model is also a model of the generated ABox. Instance checking queries are computed from the most-specific concepts (MSC) of the individuals used in the ABox. The maximum level we were able to generate for each problem is indicated in Table 1 (see the column **max**).

3.2 Evaluating Optimization Strategies

For evaluating the performance of RACE, a comparison with other systems might be in order. However, there are only a few sound and complete systems that can deal with ABox reasoning. For instance, $\mathcal{KRIS}$ cannot solve the instance checking problems mentioned in Table 1 even at the first level (the same is true for the DL'98 ABox benchmarks). ABox reasoning in general and realization in particular was not the main design goal of the $\mathcal{KRIS}$ architecture. To the best of our knowledge, currently, there is no other implemented system described in the literature, that can compete with RACE's ABox reasoning subsystem in terms of expressiveness of the logic and average-case performance of the inference algorithms. So, in order to document the performance of RACE we explain the influence of optimization strategies concerning the level of the hardest problem that can be solved by RACE within a time limit.

In Table 1 runtimes for the nine benchmark problem sets described above are presented. For all tests we have used Macintosh Common Lisp 4.2 (80MB memory assignment) on a 266MHz G3 Macintosh Powerbook computer. A timeout is set to 500 seconds, i.e. if the associated TBox can be classified within 500 seconds, there are additional 500 seconds for processing the ABox assertions and solving the instance checking problems.

In the columns in Table 1 we have listed the results for running the benchmarks in different modes offered by the RACE architecture. We use the labels **i**, **r**, **c**, **d**, **p**, for marking the running mode for the benchmarks (see Table 1). The letter **i** is used for instance checking only, **r** is used for solving the problems with realization. The letters **c**, **d**, and **p** indicate optimization strategies (see below). In the third column of Table 1 (labeled **icdp 1**) we list the runtimes (in seconds) of the first level inference problems with all possible optimizations enabled. In the next two columns (labeled **icdp** and **rcdp**, respectively) two subcolumns are used to present the maximal level and the runtime for the problem of the maximal level. As could be expected, in the realization mode, less problems can be solved. Note that, in principle, it is not necessary to first realize the ABox in order to solve the instance checking problems. Realization is done only for the performance evaluation. The columns **icdp** and **rcdp** indicate that RACE is more than two orders of magnitude faster than previous systems when all optimizations are enabled. In the following paragraphs the optimization techniques are discussed in detail.

3.3 Maximizing the Effect of Caching

In order to make realization as fast as possible we investigated ways to maximize the effect of caching techniques supplied by RACE's ABox consistency checking architecture.

The idea is the following. Let us assume, the MSC of a certain individual i has to be computed. We transform the original ABox in such a way that acyclic "chains" of roles and individuals are represented by an appropriate exists restriction (see below for a formal definition of the transformation rules). The corresponding concept and role assertions representing the chains are deleted from the ABox. We illustrate this contraction idea by an example presented in Figure 1. The individual a to be realized is assumed to be represented by a gray circle. An exists restriction "representing" an eliminated chain is added as a concept assertion to the individual starting the eliminated chain. In the presence of number restrictions, the contraction is not applied if in the original ABox there is more than one filler on the right-hand side of a certain role. The reason

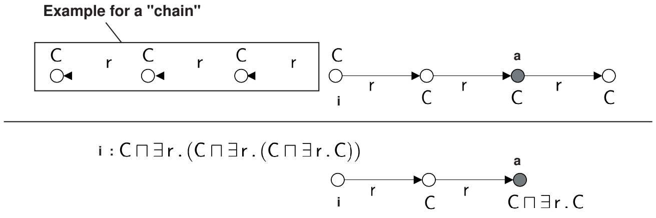

Fig. 1. Example for ABox chain contraction (see text). The upper ABox is transformed into the lower one.

is that individuals explicitly mentioned in the ABox must not be merged (unique name assumption). If these non-mergable individuals are "represented" by exists restrictions, this non-merge information is lost. It can be shown that a transformed contracted ABox is (in)consistent iff the original ABox is (in)consistent. Let a be the individual that is realized (i.e. after the transformation, constraints of the form $a : \neg C$ are added).

RC1: Contraction Rule for $\mathcal{ALC}$.
Premise: $(i, j) : r \in A, j : C_1 \in A, \ldots, j : C_n \in A, i \neq a,$
$\neg\exists(j, k) : r' \in A,$
$\neg\exists(l, j) : r'' \in A, l \neq i,$
$\neg\exists j : C_{n+1} \in A : \forall i \in \{1, \ldots, n\} : C_{n+1} \neq C_i,$
Consequence: $A' = (A \setminus \{(i, j) : r, j : C_1, \ldots, j : C_n\}) \cup$
$\{i : \exists r.C_1 \sqcap \ldots \sqcap C_n\}$

We define an algorithm *contraction_alc*(A) that recursively applies the rule **RC1** to a consistent input ABox A as long as possible. The computed ABox A' is used as A in subsequent steps. Finally, when **RC1** is no longer applicable, the algorithm returns the ABox A'.

Proposition 1. *The algorithm* contraction_alc*(A) terminates and, given A is an $\mathcal{ALC}$ ABox, the ABox A' being returned is consistent iff the original ABox A is consistent.*

Proof. (Sketch) In every step **RC1** removes a role constraint of the form $(i, j) : r$. In the premise of **RC1** a role constraint is used as a precondition for applying the rule. Thus, if there are n role constraints in the original ABox, the algorithm *contraction_alc*(A) terminates after n steps, at the latest. It is easy to that, if the rule **RC1** is applied, the ABox A' does not contain constraints for the individual j. However, considering the new exists constraint for i and the sound and complete rules of the underlying tableaux calculus [3] we can see that the tableaux rules will create a new individual j' with the same constraints as for j in the original ABox. The additional constraint $j' : C_1 \sqcap \ldots \sqcap C_n$ is expanded into $j' : C_1, \ldots, j' : C_n$. □

The contraction rule **RC1** can be used for $\mathcal{ALC}$ ABoxes (whose associated TBoxes are also $\mathcal{ALC}$ TBoxes). For $\mathcal{ALCNH}_{R^+}$ ABoxes we need a more conservative contraction rule **RC2**, which is defined as follows (a is the individual being realized, the predicate $csr(r, s)$ is true for all roles r and s which have a common superrole or are equal).

RC2: Contraction Rule for $\mathcal{ALCNH}_{R^+}$.
Premise: $(i, j) : r \in A, j : C_1 \in A, \ldots, j : C_n \in A, i \neq a,$
$\neg\exists(j, k) : r' \in A,$
$\neg\exists(l, j) : r'' \in A, l \neq i,$
$\neg\exists(i, o) : s \in A, o \neq j, csr(r, s),$
$\neg\exists j : C_{n+1} \in A : \forall i \in \{1, \ldots, n\} : C_{n+1} \neq C_i,$
Consequence: $A' = (A \setminus \{(i, j) : r, j : C_1, \ldots, j : C_n\}) \cup$
$\{i : \exists r.C_1 \sqcap \ldots \sqcap C_n\}$

In the same way as above, we define an algorithm *contraction_alcnhr+*(A) that recursively applies **RC2** to a consistent input ABox A.

Proposition 2. *The algorithm* contraction_alcnhr+*(A) terminates and, given A is an $\mathcal{ALCNH}_{R^+}$ ABox, the ABox A' being returned is consistent iff the original ABox A is consistent.*

Proof. (Sketch) Given the proof sketch for *contraction_alc*(A) it easy to see that the algorithm terminates and is sound and complete. The additional constraint in the premise guarantees that role constraints are not transformed into appropriate exists constraints if there is more than one role constraint $(i, j) : r$ with i being used more than once on the left-hand side of a role constraint for r. If there exists a common superrole, number restriction might be imposed. □

We have seen that transforming the ABox with one of the appropriate above-mentioned algorithms does not change anything from a theoretical point of view. From a practical point of view, the transformation has advantages. For implementing the realization inference service for an individual a we can safely compute a contracted ABox A', add $a : \neg C_i$ with C_i iterating over the set of atomic concepts used in the TBox. The individual a is an instance of C_i if the resulting ABox $A' \cup \{a : \neg C_i\}$ is inconsistent.

In the RACE system some optimizations are implemented for dealing with simple cases. Since we compute the direct subsumers and the told subsumers beforehand, we apply the same optimization techniques as introduced in [1]. However, in general, an "expensive" tableaux proof is required for solving an instance checking problem. By taking the contracted ABox instead of the original one, RACE can profit from the optimizations for ∃-constraints introduced in Section 2.1. Caching and model merging strategies are exploited for checking the satisfiability of transformed ABoxes in different instance checking calls during the TBox traversal phase of the realization algorithm. In addition, the number of different individuals to be treated is reduced.

Our empirical tests indicate that chain contraction can be effectively implemented. Due to our experiences, the set of ABox assertions is reduced by several orders of magnitude. The speed gain of this technique is indicated in Table 1. It is documented with the runtimes in the double-column **rdp** (see Table 1, a missing **c** indicates that chain contraction is disabled).

3.4 Propagation of Value Restrictions

If there are role assertions in an ABox, it might be possible to propagate value restrictions to appropriate role fillers in order to collect additional told subsumers as an auxiliary preprocessing step. The last double-column **rcd** of Table 1 presents the effect of propagating value restrictions. For the ABoxes used in the benchmarks, computing told subsumers by propagation of value restrictions is not very effective. Compared to the column for **rcdp** neither a performance gain nor a performance loss is indicated.

In many cases, for complex ABoxes it is faster to first partition the ABox into different parts that do not "influence" one another. It should be noted that for large ABoxes, the partitioning overhead might not be tolerable (auto-partitioning can be disabled in RACE). RACE also supports the management of several user-defined ABoxes (with respect to a single terminology).

3.5 Deep Model Merging and TBox Reasoning

In order to more extensively document the effect of the "deep model merging" optimization strategy, we evaluate RACE's TBox reasoning subsystem which is based on [1] and [4]. In Table 2 we have listed the runtimes for classifying realistic TBoxes from the DL'98 benchmarks with quite many terminological axioms (see the column named "Conc's", for details see [5]). The tests have been carried out for all systems, RACE, FaCT and $\mathcal{KRIS}$ using the same test environment as indicated above. We used version 1.11 of FaCT which does not support number restrictions (FaCT can handle $\mathcal{ALCH}f_{R^+}$ TBoxes). Furthermore, $\mathcal{KRIS}$ does not support GCIs, so some tests are left out (indicated with NA). The columns RACE and RACE **d** indicate the runtimes of RACE without deep model merging and with deep model merging enabled, respectively. For the *ABox* problems being considered above, enabling or disabling deep model merging has no effect.

Table 2. Realistic TBox classification tests (times in secs).

KB	Conc's	Time			
		RACE	RACE **d**	FACT	$\mathcal{KRIS}$
ckb-roles	79	0.13	0.15	NA	0.28
ckb GCIs	79	13.70	13.94	NA	NA
datamont-roles	120	0.38	0.21	NA	0.53
espr-roles	142	0.19	0.22	NA	0.41
espr GCIs	142	12.74	12.53	NA	NA
fss-roles	132	0.34	0.50	NA	0.67
fss GCIs	132	23.25	23.55	NA	NA
wisber-roles	140	0.35	0.36	NA	0.69
wisber GCIs	140	4.81	4.43	NA	NA
galen GCIs	2748	125.95	74.57	97.18	NA
galen1	2728	56.84	38.88	43.07	>2000
galen2	3926	32.39	27.80	25.93	189.97

4 Conclusion

An increase in performance can be achieved by algorithms that are based on informed search (together with appropriate index structures for accessing necessary information). There is no single technique which is a panacea for speeding up the inference engine. A combination of different techniques is not only required for efficient concept consistency checking and TBox reasoning but is even more important for ABox reasoning (see also [2]). We have seen that a technique such as the "chain contraction" optimization strategy is very effective in the current setting of ABox benchmarks.

For the well-known ABox reasoning problems, instance checking and realization, the theory about optimization techniques has been extended by new transformation strategies that help to make caching as effective as possible. As the evaluation results on $\mathcal{ALC}$ ABoxes indicate, realization with RACE is several orders of magnitude faster than with previous systems. The tests indicate that, in the average case, an ABox consistency checking system can also be used in places where a concept consistency tester is sufficient (e.g. for TBox reasoning). The imposed overhead of dealing with individuals is minimal and the performance of RACE with respect to TBox reasoning is comparable to "fast" TBox reasoning systems such as FaCT. In addition, RACE can deal with TBoxes based on the more expressive language $\mathcal{ALCNH}_{R^+}$ and also supports ABox reasoning.

Acknowledgments

We would like to thank E. Franconi, I. Horrocks and P. Patel-Schneider for valuable discussions.

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. V. Haarslev and R. Möller. Applying an $\mathcal{ALC}$ ABox consistency tester to modal logic SAT problems. In N. Murray, editor, *Proceedings TABLEAUX'99, The 6th Int. Conf. on Theorem Proving with Analytic Tableaux and Related Methods*, pages 24–28. Springer Verlag, Berlin, 1999.
3. V. Haarslev and R. Möller. Expressive ABox reasoning with number restrictions, role hierarchies, and transitively closed roles. Technical report, University of Hamburg, Computer Science Department, 1999. In preparation.
4. I. Horrocks and P. Patel-Schneider. Optimising description logic subsumption. *Journal of Logic and Computation*, 9(3), 1999.
5. I. Horrocks and P.F. Patel-Schneider. DL systems comparison. In *Proceedings of DL'98, International Workshop on Description Logics*, pages 55–57, Trento(Italy), 1998.