Handling Boolean ABoxes

Carlos Areces
INRIA–Lorraine, Nancy, France
areces@loria.fr

Patrick Blackburn
INRIA–Lorraine, Nancy, France
patrick@aplog.org

Bernadette Martinez Hernandez
Fac. de Cs. de la Comp. Benemérita Univ. Autónoma de Puebla, México
titab@solarium.cs.buap.mx

Maarten Marx
LIT, ILLC, Universiteit van Amsterdam, The Netherlands
marx@science.uva.nl*

Abstract

We consider description logic knowledge bases in which the ABox can contain Boolean combinations of traditional ABox assertions (represented as clauses or sequents). A linear reduction of such knowledge bases into a standard format (allowing only conjunctive assertions) is described which preserves knowledge base satisfiability. Similar results are presented for Boolean TBoxes and Boolean combinations of both ABox and TBox statements.

1 Introduction and Motivation

A description logic (DL) ABox has some resemblance to a relational database with only unary and binary relations. But, in contrast to the database model, the knowledge contained in the ABox is not assumed to be complete; it only describes a partial model of the world. This partiality is not only due to lack of information (absence of $i{:}C$ in the ABox does not mean that $i{:}C$ is false) but also to the presence of disjunctive properties like $i{:}C \sqcup D$. Thus partial knowledge about the world can be represented in the ABox, but only as a conjunction of ABox assertions of the form $i{:}C$ and $R(i, j)$. There are several applications for which this format is too restrictive, for instance in planning and diagnosis. In these tasks one often needs to describe the partial knowledge of the world using disjunctions or other Boolean constraints.

Consider the problem of organizing a dinner. The popular Bill is attending, and thanks to his presence there are a number of constraints on the setting of the table:

- Mary hates to sit next to smokers – unless she sits next to Bill.
- If Mary sits next to Bill, then Sue will be jealous if she doesn't too.

*Research supported by NWO grants 612-062-001 and 612.000.106; and the INRIA funded LIT-LED Research Alliance.

We can state these constraints as propositional combinations of ABox assertions:

$$\texttt{SitNextTo}(\texttt{mary},\texttt{bill}) \vee \texttt{mary}{:}\forall\texttt{SitNextTo}.\neg\texttt{SMOKER}$$
$$\texttt{SitNextTo}(\texttt{mary},\texttt{bill}) \rightarrow (\neg\texttt{SitNextTo}(\texttt{sue},\texttt{bill}) \rightarrow \texttt{sue}{:}\texttt{JEALOUS}).$$

Such statements involve Boolean combinations of ABox assertions. But standard DLs (and, more crucially from a computational point of view, available DL reasoners) allow only a *conjunction* of assertions in the ABox. We will consider Boolean ABoxes and use already available provers to work with them.

Satisfaction of a Boolean ABox by an interpretation is defined by extending the definition from assertions to propositional functions over assertions. We consider the key DL inference problem of knowledge base (KB) satisfaction: given a TBox T and an Boolean ABox B, does (T, B) have a model? As is well known, given the expressive power of $\mathcal{ALC}$, all desirable reasoning services in a DL system can be reduced to this. Now, the satisfiability of a KB with a Boolean Abox (T, B) can obviously be reduced to ordinary (TBox, ABox) satisfaction by non-deterministically choosing a (propositional) valuation which satisfies B, write this as an ABox A, and check whether (T, A) has a model. But implementing such a non-deterministic approach would lead to a search through all the possible valuations. In this paper we will investigate an alternative: given the expressive power provided by standard DL provers like RACER [3][1], we develop a linear transformation from a pair (T, B) (where B will be assume to be in clausal or sequent form) into an equisatisfiable standard KB (T', A) which can be directly checked for satisfiability. We then show that a similar strategy can be used to handle Boolean TBoxes, and what we call Interaction Boxes, which contain both ABox and TBox statements.

2 Translating Boolean ABoxes

In this section we show that the KB satisfiability problem for Boolean ABoxes can be reduced to the satisfiability problem of a traditional KB (possibly in an extended language).

Definition 1 An *ABox atom* is a formula of the form $i{:}C$ or $R(i, j)$, for C a (complex) concept and R a relation. An *ABox literal* is an ABox atom or its negation. A *clause* is a set of ABox literals. A *unit clause* is a singleton containing a positive literal. A *Boolean ABox* is a set of clauses.

A model $\mathfrak{M}$ satisfies a clause $\{l_1, \ldots, l_n\}$, if $\mathfrak{M} \models l_i$, for one of the literals. $\mathfrak{M}$ satisfies a Boolean ABox if it satisfies all its clauses.

As we see in Definition 1, we will represent Boolean ABoxes as sets of clauses (equivalently, we could have used sequents or ordinary disjunctions). The linear time reduction is defined given such an input[2].

[1] To be precise, we need the expressivity of $\mathcal{ALC}$ with either functional roles and role conjunction, or role hierarchies and qualified number restrictions.

[2] Of course, an exponential blowup can occur while moving an arbitrary Boolean ABox into clausal or sequent form, but we won't discuss this issue here. In any case we believe that a (say) sequent-style representation of Boolean ABoxes is conceptually appropriate and does not impose expressive limitations on the user; on the contrary it can actually help her to properly structure the information.

For the reduction, we have to rewrite a set of clauses into a set of unit clauses. Let us consider all possible problematic cases: 1) occurrences of $\neg i{:}C$; 2) clauses containing $i{:}C$ and $i{:}D$; 3) clauses containing $i{:}C$ and $j{:}D$; and 4) clauses containing $R(i,j)$ or its negation.

We will now go trough all these cases step by step and see how they can be handled. The first two cases are easy because for any model $\mathfrak{M}$ it holds that $\mathfrak{M} \models \neg i{:}C$ iff $\mathfrak{M} \models i{:}\neg C$ and $\mathfrak{M} \models i{:}C \vee i{:}D$ iff $\mathfrak{M} \models i{:}C \sqcup D$. So we can replace each occurrence of a literal $\neg i{:}C$ by a literal $i{:}\neg C$ without change of meaning. Additionally clauses containing several literals of the form $i{:}C$ for some i can be reduced to clauses containing just one occurrence.

To solve the third problem we extend the language. Instead of a clause $i{:}C \vee j{:}D$, we consider the two unit clauses $R(i,j)$ and $i{:}(C \sqcup \forall R.D)$ in which R is a new relation symbol. We claim that for arbitrary clauses Cl_i,

$$\begin{array}{l}\text{For any KB, KB } \cup\{\{Cl_1,\ldots,Cl_n,i{:}C,j{:}D\}\} \text{ is satisfiable if and only if}\\ \text{KB } \cup\{\{R(i,j)\},\{Cl_1,\ldots,Cl_n,i{:}(C \sqcup \forall R.D)\}\} \text{ is satisfiable.}\end{array} \quad (1)$$

Proof. [$\Rightarrow$] Let $\mathfrak{M}$ satisfy KB $\cup\{\{Cl_1,\ldots,Cl_n,i{:}C,j{:}D\}\}$. Expand $\mathfrak{M}$ to $\mathfrak{M}'$ by setting $I'(R) = \{(i,j)\}$ and $I' = I$ for all other symbols. Then $\mathfrak{M}' \models KB$, $\mathfrak{M}' \models R(i,j)$, and if $\mathfrak{M} \models i{:}C \vee j{:}D$ then $\mathfrak{M}' \models i{:}C \sqcup \forall R.D$, as desired.

[$\Leftarrow$] Let $\mathfrak{M} \models KB \cup \{\{R(i,j)\},\{Cl_1,\ldots,Cl_n,i{:}C \sqcup \forall R.D\}\}$. We only need to consider the case that $\mathfrak{M} \models i{:}C \sqcup \forall R.D$. Then either $\mathfrak{M} \models i{:}C$ or $\mathfrak{M} \models i{:}\forall R.D$. In the second case, since $\mathfrak{M} \models R(i,j)$, $\mathfrak{M} \models j{:}D$. Whence $\mathfrak{M}$ is a model for KB $\cup\{\{Cl_1,\ldots,Cl_n,i{:}C,j{:}D\}\}$. QED

The last problem is the most difficult to solve. We need to reduce clauses like $i{:}C \vee R(j,k)$ and $\neg R(i,j) \vee R(j,k)$ to unit clauses. We do this by uniformly replacing every occurrence of $R(i,j)$ by an equivalent formula of the form $i{:}C$. By the previous results, any set of clauses containing no relational assertions can be transformed to an equisatisfiable set of unit clauses.

The idea is best explained using role conjunction. Replace every occurrence of $R(i,j)$ by $i{:}\exists R \sqcap R'.\top$, in which R' is a new functional role symbol, and add the unit clause $R'(i,j)$ to the KB. Obviously for functional R' it holds that $R'(i,j) \models R(i,j) \equiv i{:}\exists R \sqcap R'.\top$.

Unfortunately, standard DL languages do not allow role conjunction and we have to make a more involved encoding. We do the following: let A be a set of clauses in which $R(i,j)$ occurs negatively or in non unit clauses. Let *FakeR* and *IamJ* be a new relation and concept symbol, respectively. Consider the following KB. Note that $(*1)$ is a TBox statement, the others are ABox assertions.

$$(*) = \left\{\begin{array}{ll}(*1) & R \sqsubseteq \mathit{FakeR}\\ (*2) & \mathit{FakeR}(i,j)\\ (*3) & j{:}\mathit{IamJ}\\ (*4) & i{:}\exists_{\leq 1}\mathit{FakeR}.\mathit{IamJ}.\end{array}\right.$$

We claim that

$$(*) \models R(i,j) \equiv i{:}\exists R.\mathit{IamJ}. \quad (2)$$

This is most easily proved by a tableau-like argument as follows.

From left to right:

1)	$R(i,j)$	
2)	$\neg i{:}\exists R.IamJ$	
3)	$\neg j{:}IamJ$	($\forall$ rule on 1 and 2)
4)	$j{:}IamJ$	(premise $*3$)
	$\bot$	(by 3, 4)

From right to left

1)	$i{:}\exists R.IamJ$	
2)	$\neg R(i,j)$	
3)	$R(i,k)$	($\exists$ rule on 1)
4)	$k{:}IamJ$	
5)	$FakeR(i,k)$	(by 3 and $*1$)
6)	$FakeR(i,j)$	($*2$)
7)	$j{:}IamJ$	($*3$)
8)	$k = j$	(by 4, 5, 6, 7 and $*4$)
	$\bot$	(by 2, 3, 8)

Now given a Boolean KB (T, B) with $R(i,j)$ occurring negatively or in non unit clauses in the ABox, let KB' be $(*) \cup T \cup B[R(i,j)/i{:}\exists R.IamJ]$. Here $B[\phi/\psi]$ denotes the result of uniformly replacing ϕ by ψ in B. We claim that

$$(T, B) \text{ is satisfiable iff } (*) \cup T \cup B[R(i,j)/i{:}\exists R.IamJ] \text{ is satisfiable.} \tag{3}$$

The direction from right to left follows immediately from (2). For the other direction, let $\mathfrak{M} \models (T, B)$. Expand $\mathfrak{M}$ to $\mathfrak{M}'$ by setting $I'(IamJ) = \{j\}$ and $'(FakeR) = I(R) \cup \{(i,j)\}$ and let I' be equal to I for all other symbols. Obviously $\mathfrak{M}' \models (*)$ and still $\mathfrak{M}' \models (T, B)$. Whence, by (2), $\mathfrak{M}' \models B[R(i,j)/i{:}\exists R.IamJ]$.

With this we have covered all problematic cases. The rewrite algorithm is given below:

Input: a Boolean KB (T, B)

- if $R(i,j)$ occurs negatively or in non unit clauses, then uniformly replace $R(i,j)$ in A by $i{:}\exists R.IamJ$, and add the formulas in $(*)$ to the KB (using new symbols every time).
  Intermediate result: a Boolean ABox without relational assertions.
- push negations inside using the validity $\neg i{:}C \equiv i{:}\neg C$.
  Intermediate result: a Boolean ABox without negative literals.
- collect disjunctions concerning the same terms using the validity $i{:}C \vee i{:}D \equiv i{:}C \sqcup D$.
  Intermediate result: a Boolean ABox in which every clause contains at most one literal $i{:}C$ for each i.
- repeatedly transform disjunctions concerning different terms as in (1).

Output: a standard KB (T', A).

3 Boolean T-Boxes and Interaction Boxes

As the following examples show, it can also be useful to handle Boolean combinations of TBox definitions, or even a Boolean mixture of TBox and ABox statements (knowledge bases allowing such combinations have already been investigated, for example in the work of Wolter and Zakharyaschev [7, 8]).

TBox: A DL TBox consists of a set of definitions of the form $C \sqsubseteq D$. It might be that in our modeling domain our definitional knowledge is only partial. For example, we might know that the *pulis* are a subspecies of either cats or dogs, but we do not know which one. This is then expressed as $(\texttt{PULI} \sqsubseteq \texttt{CAT}) \vee (\texttt{PULI} \sqsubseteq \texttt{DOG})$. Note that this is *not* equivalent to $\texttt{PULI} \sqsubseteq \texttt{CAT} \sqcup \texttt{DOG}$. From $(\texttt{PULI} \sqsubseteq \texttt{CAT}) \vee (\texttt{PULI} \sqsubseteq \texttt{DOG})$ and $\texttt{CAT} \sqcap \texttt{DOG} \sqsubseteq \bot$ together with the ABox $\{i{:}\texttt{PULI}, i{:}\texttt{CAT}, j{:}\texttt{PULI}\}$ it follows that $j{:}\texttt{CAT}$. That is, if there is a puli which is a cat, then all pulis are cats. We will call such a Boolean combination of TBox definitions a *Boolean TBox*.

IBox: A Boolean *IBox* (for *Interaction Box*) is a Boolean combination of *both* ABox assertions and TBox definitions. E.g., $\texttt{snoopy}{:}\texttt{PULI} \Rightarrow \texttt{PULI} \sqsubseteq \texttt{DOG}$ expresses that if we know that snoopy is a puli, then every puli is a dog.

There is another way in which these mixed statements are useful. For instance, the conjunctive query $\exists x(\text{snoopy}{:}\texttt{PULI} \wedge x{:}\texttt{PULI} \wedge x{:}\neg\texttt{DOG})$ is true given a knowledge base KB if and only if the knowledge base consisting of KB together with $\texttt{snoopy}{:}\texttt{PULI} \Rightarrow \texttt{PULI} \sqsubseteq \texttt{DOG}$ is not satisfiable. So these interaction statements naturally occur when reducing the answering of conjunctive queries to the (well understood) knowledge base satisfiability problem [6, 5]. More on queries below.

Translating Boolean TBoxes is fairly straightforward (and follows directly from the literature) if we have transitive roles and role hierarchies, because in this case the TBox SAT problem can be reduced to the concept SAT problem for the empty TBox (see [1]).

Finally consider the following example of an interaction box: $\texttt{mary}{:}\texttt{AtDinner} \Rightarrow (\texttt{GUEST} \sqsubseteq \neg\texttt{SMOKER})$. Using R for our "universal role" as described above we rewrite this formula as $(\texttt{mary}{:}\texttt{AtDinner}) \Rightarrow \texttt{root}{:}\forall R.(\neg\texttt{GUEST} \sqcup \neg\texttt{SMOKER})$, which can be rewritten as the clauses $\{\texttt{NewR}(\texttt{mary}, \texttt{root})\}$, $\{\texttt{mary}{:}(\neg\texttt{AtDinner} \sqcup \forall\texttt{NewR}.\forall R.\neg\texttt{GUEST} \sqcup \neg\texttt{SMOKER})\}$ by the techniques described in Section 2.

Queries. The problem of querying DL knowledge bases is an active area of research. This inference service is somewhat orthogonal to the usual DL inference tasks of subsumption and satisfiability checks. The problem is that there are no simple reductions of a query task to the well understood KB satisfiability task. Consider the grounded Boolean query $R(i, j) \sqcap j{:}C$ to the KB (T, A). The answer will be yes if and only if $(T, A) \cup \{\neg(R(i, j) \sqcap j{:}C)\}$ is not satisfiable. But unfortunately, $\neg(R(i, j) \sqcap j{:}C)$ is not in the DL ABox format. But it is a Boolean ABox, so with the reduction presented in the previous section the query to the KB can be reduced to ordinary KB satisfiability.

Unfortunately this strategy breaks down for queries containing variables. Here often much more elaborate mechanisms are needed, as described for instance in [6]. But in some cases, a simple reduction using Boolean IBoxes is possible. Consider the following three closed conjunctive queries:

$$\exists x \exists y(x = i \wedge y = j \wedge x{:}C \wedge y{:}D) \tag{4}$$

$$\exists x \exists y(y = j \wedge x{:}C \wedge y{:}D) \tag{5}$$

$$\exists x \exists y(x{:}C \wedge y{:}D) \tag{6}$$

To answer whether (4) follows from a KB, we add the Boolean ABox $(i{:}\neg C \vee j{:}\neg D)$ to it and check its satisfiability. To settle (5), which is equivalent to $j{:}D \wedge \exists x x{:}C$, we

should add the IBox $j{:}\neg D \vee (C \sqsubseteq \bot)$ to the KB and decide whether it is satisfiable. For the last query (6), we add the Boolean TBox $(C \sqsubseteq \bot) \vee (D \sqsubseteq \bot)$ to the KB.

4 Empirical Testing and Optimizations

The encoding we described in the previous sections is simple to implement as a preprocessing step and would allow the use of Boolean KBs in a standard DL prover like RACER with no further complications.

There is ample room for optimizations though. In essence Boolean KBs add an extra source of propositional non-determinism and our reduction just hands this extra burden over to the DL prover. A better strategy might be to preprocess the input by a dedicated propositional solver. This solver will treat the complex DL assertions just as propositional atoms and hence might be quicker in reducing it to a simpler problem. Simplifications that eliminate atoms of the form $\neg R(i,j)$ would be particularly useful. Similarly, if we can delete all positive occurrences of $R(i,j)$ and only have a single clause $\{\neg R(i,j)\}$ left, this clause can be discarded leaving an equisatisfiable problem. A propositional prover could, in addition, generate propositionally satisfiable problems and feed them (sequentially or in a distributed fashion) to the DL prover (see [2]).

To obtain a (very preliminary) evaluation of the ideas discussed in this paper we implemented two prototypes `ba2racer-red` and `ba2racer-dpll` [4]. Given the space restrictions, our description of the implementation decisions and the empirical testing performed will have to be brief.

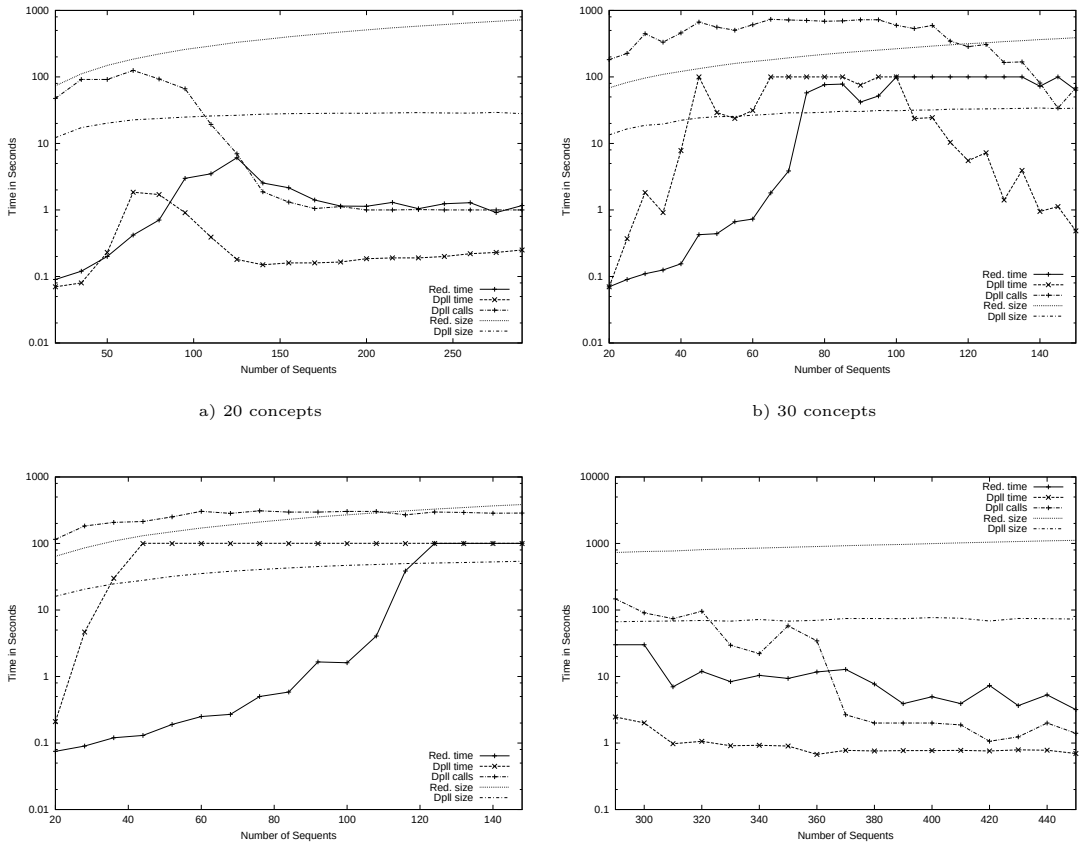
`ba2racer-red` accepts a file describing a standard (non-Boolean) TBox and a Boolean ABox specified in terms of sequents. It performs the reduction described in Section 2 generating an equisatisfiable knowledge base and calls RACER on this single file.

`ba2racer-dpll` accepts the same input as above, but implements a simple Davis-Putnam algorithm on the Boolean ABox (considering different atomic assertions in the Boolean ABox as different propositional symbols). For each consistent propositional assignment found, a standard KB is generated and RACER is called on this file. The input KB is unsatisfiable if each of the possible propositional assignments fails to produce a satisfiable KB.

We also implemented a simple random generator of Boolean ABoxes. This generator starts by randomly creating a satisfiable set of propositional sequents. Then it randomly assigns ABox assertions to each of the atomic symbols. Different parameters are used to define the number of sequents generated, number of available atomic concepts and roles, maximal modal depth, etc.

In the graphs presented in Figure 1 we plot CPU time (in log-scale) against the number of sequents in the Boolean knowledge base. In each data point we plot mean time of 80 independent tests, with a time-out of 100 seconds. During random generation we fixed most of the parameters. The only dimension we explored is the effect that the size of the signature (more precisely, the number of atomic concepts available for generating ABox assertions) has in the behavior of the different methods. We adjusted the number of sequents generated to ensure exploration of the spectrum from trivially satisfiable to trivially unsatisfiable[3]. The other relevant, fixed parameters

[3]Graphs c) and d) are actually two fragments of the same test set. For this configuration, the



a) 20 concepts

b) 30 concepts

c) 70 concepts sat. part

d) 70 concepts unsat. part

Figure 1: Testing `ba2racer-red` and `ba2racer-dpll`.

were: number of roles in the signature = 2, modal depth = 2, length of a sequent = 3, probability of generating a modal formula = 0.5.

Red. Time and *Dpll Time* plot the mean time required by the reduction and the dpll based approach respectively. *Red. Size* and *Dpll Size* plot the average size (in lines) of the KB generated while *Dpll Calls* shows the average number of calls to RACER (i.e., propositionally consistent assignments found).

The preliminary testing seems to indicate different behaviors for the two approaches. `ba2racer-red` seems to perform best on satisfiable knowledge bases, specially over a large signature. In this cases the number of satisfiable propositional assignments is big, and the repetitive calls to RACER degrade `ba2racer-dpll`. Conversely, when the signature is small or when the number of clauses is big so that the probability of redundancy in the Boolean ABox is high, `ba2racer-dpll` outperforms `ba2racer-red` as in this cases the reduction generates a knowledge base which is just too hard (and big) for RACER to handle efficient. The pruning done by the propositional dpll algorithm leads to the generation of a few simple knowledge bases (even none in some cases!) that are easily checked by RACER.

critical area where satisfiability reaches maximum uncertainty was too difficult for both approaches giving too many timeouts.

5 Conclusion

We have shown how the functionality of standard DL provers like RACER can be extended to Boolean KBs. Boolean KBs were divided into three types, Boolean ABoxes, Boolean TBoxes, and what we have termed Boolean IBoxes (interaction Boxes). This seems to be a natural division, paving the way for an incremental approach to optimizations. The rewrite algorithm can also be used to answer propositional ABox queries to a DL knowledge base. We see two main directions for further research. The first concerns optimizations. Is there a better way of rewriting? How much can a Boolean preprocessor help in speeding up the problem solving process? The second concerns the scope of the query processing we can do using Boolean IBoxes. We would like to have syntactic criteria on queries which guarantee that a reduction to Boolean IBox satisfiability is possible.

References

[1] M. Buchheit, F. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1:109–138, 1993/94.

[2] F. Giunchiglia and R. Sebastiani. A sat-based decision procedure for $\mathcal{ALC}$. In L. Aiello, J. Doyle, and S. Shapiro, editors, *Proc. of the 5th International Conference on Principles of Knowledge Representation and Reasoning (KR'96)*, pages 304–314, Cambridge, USA, 1996.

[3] V. Haarslev and R. Möller. Description of the RACER system and its applications. In *Proc. of the International Workshop on Description Logics (DL-2001), Stanford, USA, 1.-3. August 2001*, pages 132–141, 2001.

[4] B. Martínez Hernández. Automated reasoning with boolean aboxes. Master's thesis, Institute for Logic, Language and Computation, 2002.

[5] I. Horrocks, U. Sattler, S. Tessaris, and S. Tobies. How to decide query containment under constraints using a description logic. In *Logic for Programming and Automated Reasoning (LPAR 2000)*, volume 1955 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2000.

[6] S. Tessaris. *Questions and answers: reasoning and querying in Description Logic.* PhD thesis, University of Manchester, 2001.

[7] F. Wolter and M. Zakharyaschev. On the decidability of description logics with modal operators. In G. Cohn, L. Schubert, and S. Shapiro, editors, *Proc. of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR-98)*, pages 512–523. Morgan Kaufmann, 1998.

[8] F. Wolter and M. Zakharyaschev. Temporalizing description logics. In D. Gabbay and M. de Rijke, editors, *Frontiers of Combining Systems*, pages 379–402. Studies Press/Wiley, 1999.