Expressive Description Logics via SAT: The Story so Far

Francis Gasse and Volker Haarslev
Concordia University, Montreal, Quebec, Canada
{f_gasse, haarslev}@cse.concordia.ca

Abstract

The Boolean Satisfiability (SAT) problem is widely researched and the performance of such solvers largely benefit from it. Satisfiability Modulo Theory (SMT) solvers aim to leverage the good performance toward other formalisms with large propositional content. Description logics are an expressive subset of first-order logic with high complexity reasoning that could benefit from this approach. In this paper, we present a SMT-based description logic reasoner, its reasoning techniques, architecture, and some first experimental results.

1 Introduction

The Boolean Satisfiability (SAT) problem is a classical NP-complete problem studied in many sub-fields of computer science, both in theory and in practice. This sustained interest yielded substantial results over time, especially in the last few years. That newfound performance led to the idea of Satisfiability Modulo Theory (SMT) solvers, which generalize SAT solvers by adding the ability to reason over some other first-order background theory. This design was aimed to benefit from SAT solvers' excellent performances over the propositional subset of first-order logic. Furthermore, SAT solvers also perform very well with very large datasets and in parallel (e.g. [BS96]) and distributed contexts (e.g. [ZBH96]). Such contexts are usually difficult for other, more complex, decision procedures. This is another reason to look to harness SAT solvers' power outside of SAT.

Description Logics (DLs) [BCM+03] are an expressive subset of first order logic. DLs are part of the logical foundation of the Semantic Web and a de facto standard for knowledge representation in many domains, notably life sciences. Unfortunately, the average-case complexity of existing reasoning algorithms for expressive DLs often cause scalability issues. This lead us to investigate how SMT solvers, generally used for first-order theories quite different from DLs, can handle complex theories like DL and if they would retain their coveted qualities.

The remainder of this paper is structured as follows: first, we present some related work, then go through the basics of DLs and of SMT techniques. We then present our SMT-based reasoning technique, the design of its architecture and, finally, the results of some experiments we conducted to verify our hypotheses.

2 Related Work

We present some related work in order to give the reader some references w.r.t. what has been done in the field. It gives some perspective and illustrates in which ways we differ from these efforts despite some similarities. We will cover the KSAT, $K(m)2SAT$ and SAT-Tableau approaches.

KSAT [GS96] is a depth-first tableau construction procedure that uses a DPLL enumerator for the propositional part of the model building an early example of the *lazy SMT technique*. KSAT vastly outperformed all the other modal and description logic reasoners of its time. Such a domination drew attention to alternate methods and influenced even the tableau reasoners.

$K(m)2SAT$ [SV06] uses a different approach that leads to similar results. It encodes formulas of the DL $\mathcal{ALC}$ [SSS91] (which is a standard DL extending propositional logic with existential and universal quantification) into a propositional CNF problem using a technique the authors introduced. The CNF representation is then fed into a standard SAT solver. This is a textbook example of an *eager SMT design* and is thus quite monolithic.

The SAT-Tableau Calculus [KH08b, KH08a] is a SMT-based technique that is also limited to $\mathcal{ALC}$ but differs by being designed to allow extensions toward more expressive logics. It uses the abstraction paradigm to obtain propositional content. This paradigm abstracts the modal part of the theory by replacing modalities with propositional variables before verifying that the theory is propositionally satisfiable. Once it is done, the modalities are substituted to the variables to evaluate the modal satisfiability of the theory. These techniques are very similar and closely related by both their means and their results. The SAT-tableau calculus is the closest to our technique but still support a much less powerful logic.

3 Preliminaries

Before presenting our vision of a SMT-based DL system, we sketch the basics of both DLs and the SMT technique.

3.1 Satisfiability Modulo Theories

Nowadays, one often deals with very expressive and complex first-order theories for which dedicated decision procedures have been devised. Even the

most complex theories have a sizable propositional component. The classical decision procedures might not always be very appropriate. But there exist excellent tools for propositional logic, SAT solvers for instance. It might be interesting to combine SAT solvers with theory solvers ($\mathcal{T}$-solvers) and let each one handle what they do best. This is one of the main ideas behind SMT, to manage efficiently boolean reasoning with expressive theory-specific reasoning. This approach allows one to use the latest advances of SAT solving while not being limited by its relatively small expressivity.

The most popular approach to SMT, the lazy approach, is based on the integration of a SAT solver and one or many $\mathcal{T}$-solvers. The SAT solver enumerates assignments which satisfy the boolean constraints while the $\mathcal{T}$-solver checks the consistency of these assignments in $\mathcal{T}$. Building $\mathcal{T}$-solver might be time consuming but it allows one the use of data structures and algorithms best suited to the target theory, thus maximizing performance. Recent empirical evaluations have shown that the majority of the most efficient SMT implementations follow this path.

The eager approach, instead, is based on defining specialized translations to convert an input formula into an equisatisfiable propositional formula using a-priori knowledge of the theory $\mathcal{T}$. It is theoretically compatible with any theory that has a decidable ground satisfiability problem. Unfortunately, the translation is susceptible to cause an exponential blow-up of variables. Notice that such an approach is SAT solver agnostic.

3.2 The Description Logic $\mathcal{SHOQ}$

In DL individuals of a domain are considered as instances of concepts, which are organized in a hierarchy, and individuals are connected by roles, also organized in a simple hierarchy. Concepts are defined by restrictions over these roles and boolean combinations of other concepts.

A *signature* (N_C, N_R, N_I) consists of a set of *concept names* N_C, *role names* N_R, and *individual names* N_I. The set of role names contains a subset $N_{TR} \subseteq N_R$ of *transitive role names*. A *role inclusion* axiom is of the form $r \sqsubseteq s$ with $r, s \in N_R$. A *role hierarchy* $\mathcal{R}$ is a finite set of role inclusions. A role r is *simple* w.r.t. a role hierarchy $\mathcal{R}$ if there is no transitive role $s \in N_{TR}$ such that $s \sqsubseteq\!\!\!\!\!\!\cdot\;\; r$ holds.

An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$, the domain of $\mathcal{I}$, and a function $\cdot^{\mathcal{I}}$, which maps every concept name $A \in N_C$ to a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, every role name $r \in N_R$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and every individual name $a \in N_I$ to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. An interpretation $\mathcal{I}$ *satisfies* a role inclusion $r \sqsubseteq s$ if $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$, and a role hierarchy $\mathcal{R}$ if it satisfies all role inclusions in $\mathcal{R}$.

Concepts are built inductively using the following grammar, where $a \in N_I$, $A \in N_C$, $r, s \in N_R$, s is a simple role and n is a natural number:

$$C ::= \top | \bot | A | \{a\} | \neg C | C_1 \sqcap C_2 | \geq_n s.C | \exists r.C$$

We use the following standard equivalences:$C_1 \sqcup C_2 \equiv \neg(\neg C_1 \sqcap \neg C_2)$, $\forall r.C \equiv \neg(\exists r.(\neg C))$, $\leq_n s.C \equiv \neg(\geq_{n+1} s.C)$, $\forall r.C \equiv \leq_0 r.\neg C$, and $\exists r.C \equiv \geq_1 r.C$. $\mathcal{SHOQ}$'s semantics are defined as follows, where $\sharp$ denotes a set's cardinality:

$$\top = \Delta^{\mathcal{I}}, \bot = \emptyset, \{a\}^{\mathcal{I}} = \{a^{\mathcal{I}}\}, (C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}, (\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$$
$$(\forall r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} | \forall y.\langle x, y\rangle \in r^{\mathcal{I}} \; implies \; y \in C^{\mathcal{I}}\}$$
$$(\geqslant_n s.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} | \sharp\{y.\langle x, y\rangle \in s^{\mathcal{I}} and \; y \in C^{\mathcal{I}}\} \geqslant n\}$$

A *general concept inclusion* axiom (GCI) is an expression $C \sqsubseteq D$, where C and D are concepts. A finite set of GCIs is called a *TBox*. An *assertion* is an expression of the form $A(a)$, $\neg A(a)$, $r(a, b)$, $\neg r(a, b)$, or $a \neq b$ with $A \in N_C$, $r \in N_R$, and $a, b \in N_I$. An *ABox* is a finite set of assertions that can be internalized using nominals (e.g., $A(a)$ into $\{a\} \sqsubseteq A$, $r(a, b)$ into $\{a\} \sqsubseteq \exists r.\{b\}$, etc.), hence we assume w.l.o.g. that a $\mathcal{SHOQ}$ knowledge base (KB) $\mathcal{K}$ is a pair $(\mathcal{T}, \mathcal{R})$ where $\mathcal{T}$ is a TBox and $\mathcal{R}$ is a role hierarchy.

An interpretation $\mathcal{I}$ satisfies an axiom $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ and it is a model of a TBox $\mathcal{T}$ if it satisfies every axiom in $\mathcal{T}$. A concept C is called satisfiable if there exists an interpretation $\mathcal{I}$ with $C^{\mathcal{I}} \neq \emptyset$. Concept D subsumes concept C (written $C \sqsubseteq D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all models of $\mathcal{T}$. Two concepts are equivalent (written $C \equiv D$) if they are mutually subsumed. For an interpretation $\mathcal{I}$, an element $x \in \Delta^{\mathcal{I}}$ is called an instance of concept C if $x \in C^{\mathcal{I}}$. An interpretation $\mathcal{I}$ is a model of KB $\mathcal{K} = (\mathcal{T}, \mathcal{R})$ if it satisfies $\mathcal{T}$ and $\mathcal{R}$. A KB is consistent if it has a model.

4 Logical Design of a SMT-based DL reasoner

This section is the core of our vision of a SMT-based system for DL reasoning. We will describe and explain our design decisions, etc. We first present its architecture in general, then outline the reasoning process so that the theory is related with the actual reasoning process.

4.1 System's Architecture

Let us now describe the general organization of our approach, the roles of the different components as well as their interactions. For now, our technique handles concept expression with expressivity up to, and including, the $\mathcal{SHOQ}$ DL, with or without accompanying terminological axioms. Both theoretically and practically, we can distinguish three components in our architecture: the $\mathcal{T}$-solver, the SAT solver and the translator. The $\mathcal{T}$-solver is a key component since it receives the problems, dispatches the axioms and coordinates the reasoning process. The SAT solver obviously computes assignments that are valid over the propositional subset of the problem. The $\mathcal{T}$-solver then evaluates the satisfiability of these assignments w.r.t. the modal part of the theory. The translator, indeed translates DL expressions

(staying within $\mathcal{ALC}$'s expressivity) into sets of clauses, but also keeps track of the mappings between propositional variables in the SAT solver and the expressions they model in the $\mathcal{T}$-solver.

The $\mathcal{T}$-solver builds a graph that is a subset of a corresponding completion graph in a tableau reasoner. A completion graph is the result of an exhaustive application of completion rules on the DL expression to be verified. If the graph is consistent, i.e., no completion rules are applicable anymore, then this graph is an abstraction of a model of the input expression. This graph is used to verify if the assignment given by the SAT solver also satisfies the constraints that fall outside of SAT.

As usual in SMT, the information flows both ways between the SAT solver and the $\mathcal{T}$-solver, i.e., from the assignments that it verifies, the $\mathcal{T}$-solver tries to identify conflict sets (combinations of certain variable values that cannot be satisfied) and sends them to the SAT solver. That way, both part benefit from each other's strength.

4.2 Encoding $\mathcal{ALC}$ into a SAT problem

It is a well-known fact [Sch91] that $\mathcal{ALC}$ (a simple DL which includes negation, disjunction,conjunction, qualified existential and universal restrictions) is a notational variant of the modal logic K_m. We use this correspondence to adapt the previously mentioned $K(m)2SAT$ encoding [SV06] in order to encode $\mathcal{ALC}$ concept expressions into a SAT problem.

Let $\Omega()$ be an injective function which maps a pair $\langle$individual, concept expression$\rangle$ into a boolean variable. Also, if the concept expression is negated, then the variable will be as well. Given an $\mathcal{ALC}$ concept expression C, the encoder forwards the call to the function Ω as follows:

$$ALC2SAT(C) = \Omega(a_1, C) \wedge Def(a_1, C)$$

The $Def(a_i, C)$ function is recursive and unfolds the definition until all the branches reach primitive concepts. Here is the definition of this function, where the first rule matching the form of C is used:

$$\begin{array}{lcl}
A \text{ or } \neg A & : & \top \\
C_1 \sqcap C_2 & : & (\neg\Omega(a_i, C) \vee \Omega(a_i, C_2)) \wedge (\neg\Omega(a_i, C) \vee \Omega(a_i, C_1)) \wedge \\
 & & Def(a_i, C_1) \wedge Def(a_i, C_2) \\
C_1 \sqcup C_2 & : & (\neg\Omega(a_i, C) \vee \Omega(a_i, C_1) \vee \Omega(a_i, C_2)) \wedge \\
 & & Def(a_i, C_1) \wedge Def(a_i, C_2) \\
\exists r.C & : & (\neg\Omega(a_i, C) \vee \Omega(a_{i+j}, C)) \wedge Def(a_{i+j}, C) \\
\forall r.C & : & \textit{for each } \exists r.C_i \textit{ restriction applied to } a_i \textit{ do} \\
 & & (\neg\Omega(a_i, C) \vee \neg\Omega(a_i, \exists r.C_i) \vee \Omega(a_{i+j}, C_i)) \wedge Def(a_{i+j}, C_i)
\end{array}$$

The set of clauses resulting from this procedure is equisatisfiable with the original expression. This fact has been shown for K_m[SV06] and changing

the notation does not alter this property. Despite the risk of a blowup in the size of the encoded formulas, the performances of this approach are comparable with those of other tools.

4.3 $\mathcal{T}$-solver

Now we describe the $\mathcal{T}$-solver of our system. This component is responsible to verify if an assignment satisfies the logical axioms that are not expressible in $\mathcal{ALC}$, hence the axioms which cannot be rewritten into a SAT problem with the *ALC2SAT* encoding introduced above. We first give an overview of the verification process, and then detail different steps such as the graph's construction and its validation.

4.3.1 Overview of the Process

This $\mathcal{T}$-solver's verification technique is designed around a graph that closely resembles a completion graph from tableau reasoning. In this graph, nodes represent individuals, that are instances of concept(s), and the edges are roles. As we process the axioms to build the graph, we identify the axioms that are rewritable into a SAT problem (within $\mathcal{ALC}$'s expressivity) and send them to the SAT solver via the translator. The translator also keep track of a mapping from DL expressions to their variables in the SAT problems to ensure that if we get a satisfiable assignment, we only need to consider the part of the disjunctions that were assigned with a TRUE value and reciprocally, translate the $\mathcal{T}$-solver's conflict set of expressions into boolean variables. That is a consequence of the disjunctive axioms by which only one disjunct has to be satisfied thus we can ignore the other branches. The graph ultimately represents the union of all possible completion graphs for an expression, from which we only consider the parts in which variables were TRUE in the SAT problem's solution. The constricted graph, by overlooking branches set to FALSE in the current assignment, is then checked for validity w.r.t. restrictions verified by the $\mathcal{T}$-solver, and if it does not contain a clash, it is consistent and the original expression is indeed satisfiable.

4.3.2 Building the Graph

The graph building technique starts from a node labeled with the initial DL expression. We then apply a recursive technique called unfolding [BCM$^+$03] which basically replaces a concept name with the expressions it implies. Each type of expression is dealt with in a specific way that we now describe:

- $\sqcap$ Conjunctions are simply replaced by their conjuncts in the node's label.
- $\sqcup$ Disjunctions are also split with the difference that each disjunct is indexed to allow their removal if they are assigned a FALSE value by the SAT solver.

$\exists r.C$ Existential restrictions are replaced in the node's label by an at-least restriction, e.g. $\exists r.C$ is replaced with $\geq 1 r.C$. We also add an edge, labeled r, to the node along with a child node labeled C.

$\forall r.C$ Universal restrictions are processed differently w.r.t. the context. If the node has s-children, such that $s \sqsubseteq r$, then the label C is added to all these children; else if the node contains a label of the form $\geq s.C, \leq s.C$ or $\exists s.C$, the restriction is maintained. Otherwise, the restriction is removed. The restriction cannot be removed after applying it to the children since it is needed to process the role hierarchy.

$\geq_n r.C$ At-least restrictions are kept as labels and, for a restriction $\geq_n r.C$, we add the label $\exists r.C$ to let the SAT solver verify if at least one such edge is satisfiable before checking for n.

$\leq_n r.C$ At-most restrictions do not require action at this point.

Nominals Let a nominal $\{a\} \equiv N$, every such nominal is treated as a regular concept, i.e. $C \equiv \{a\}$ becomes $A \sqsubseteq C$ to allow the SAT solver to take it in consideration.

However, this technique, if applied as described, might not terminate. Indeed, consider the following lexically cyclic expression, $C \sqsubseteq \exists r.C$. Such expressions generate an infinite graph. To prevent an infinite unfolding we have to detect cyclic definitions and stop their unraveling. In tableaux reasoning, *blocking techniques* are used to stop the unfolding in such cases. The similarities between our graph and tableaux allow us to re-use these techniques. For the sake of simplicity, we will describe a simpler, less efficient technique than the one we actually use. The *subset blocking* technique simply requires that if a node's label is a subset of or equal to one of its ancestor's label, then we stop the unfolding of this node. This technique alone guarantees termination of the unfolding.

4.3.3 Checking the Graph's Validity

Checking the graph's validity is probably the most important and critical operation in our system. Indeed, being in a SMT context, when the data reaches the $\mathcal{T}$-solver, the propositional content has been validated by the SAT solver thus only the axioms not included in $\mathcal{ALC}$ are left to be verified.

To describe our graph validation technique, we will go through all the expressive features not yet verified, or not completely so, and explain how the system ensures that the assignment satisfies them. For all the expressive features below, we assume that an assignment has been computed by the SAT solver and the graph has been set up accordingly, i.e., the graph only contains the information backed up by a positive assignment from the SAT solver. Also, if any of these verifications fail, the next assignment is fetched from the SAT solver and the verification restarts with this input.

Nominals ($\mathcal{O}$) When building the graph, nominals were treated as regular concepts. We now must ensure that their semantics are respected. Due to nominals being singletons, we have to verify that the graph contains at most one instance of a nominal by merging all its instances into a single node. It is to be noted that this technique (along with the number restrictions rule) prevents the following classic problem example $A \equiv \{a_1\} \cup \{a_2\}$, $a_1 \neq a_2$, $\geq_3 R.A$ to be satisfiable. Indeed, the at-least rule would create 3 edges and nodes but the nominals rule would merge them in at most 2 nodes, thus not satisfying the restriction.

Role Hierarchies ($\mathcal{H}$) Three types of expressions can interact with the role hierarchy to cause a clash: the qualified number restrictions, the transitive roles and the universal role restrictions. The former two being handled by their own rules, we are only concerned by the universal role restrictions.

The single clash-inducing scenario is as follows: let $r \sqsubseteq s$ and $C_1 \sqcap C_2 \sqsubseteq \bot$, a node containing the labels $\forall s.C_1, \exists r.C_2$ is unsatisfiable. However, the case where $s \sqsubseteq r$ is satisfiable.

Qualified Number Restrictions ($\mathcal{Q}$) The only nodes that are of any interest when verifying number restrictions are those with at-most restrictions since an at-least alone is trivially satisfiable. For example, let us have the restriction $\leq_5 r.\top$ in a node, if the sum of at-least restrictions for role r and all its sub-roles is larger than 5, then more processing needs to be done. Otherwise, it is trivially satisfiable.

When faced with too many edges w.r.t. a qualified cardinality restrictions, we have to reduce the number of edges. Obviously, edges cannot simply be removed from the graph, thus we have to merge edges (and the attached nodes). To ensure that the graph is satisfiable, we go through the possible sets of merges that result in a correct number of roles until we find one that do not introduce a contradiction in the nodes' labels in the process.

Transitivity Since we have blocked the infinite unfolding of nodes and since $\mathcal{SHOQ}$ does not include inverse roles, we are assured that that every path in the graph is finite. Nominals allow the definition of cycle-like constructs, but without inverse roles the paths are stuck at a nominal node. To enforce the transitivity of a role, we simply need to navigate each of these paths and if two successive edges are labeled as r, s, such that $r \sqsubseteq s$, we add a s edge between the ancestor and the descendant.

4.4 SAT Solver

Following our design decisions, we consider the SAT solver as a black box that produces assignments for the $\mathcal{T}$-solver and consumes the conflict sets the $\mathcal{T}$-solver generates. This cycle keeps going until the first of two things

happens: the $\mathcal{T}$ solver accepts an assignment, meaning that this assignment also satisfies the background theory, or all possible assignments for the clauses have been generated. These events mean respectively that the concept expression is satisfiable or not.

5 Prototype Evaluation

In order to verify our hypothesis about the viability and possible advantages of this reasoning technique, we implemented a prototype in Java and tested it. For the prototype we evaluated three solvers, all open-source and Java-based, that all performed rather well. The forerunner at this point is SAT4J[1] for its API, its support and the fact that it is still being actively developed. The two other tools we evaluated are MXC[2] and SATzilla[3]. Our investigation shows that both have the potential to serve viable alternatives, thus avoiding to become too dependent on SAT4J.

At this point in our research, we are less looking for definitive and conclusive results than for a confirmation of the well-foundedness of our work. In order to do so, we chose to compare the computation times required to verify the satisfiability of the concepts in two different ontologies, one expressed in $\mathcal{ALC}$ and the other one in $\mathcal{SHON}$. The test was performed with our reasoner and the DL reasoner Pellet[4]. We settled for this small dataset following the lack of interesting $\mathcal{SHOQ}$ ontologies available. Also, we preferred to avoid artificial (generated) ontologies because they could easily introduce a bias. For the $\mathcal{ALC}$ test we used a bio-chemistry ontology,[5] where Pellet and our SMT solver took roughly the same to verify the satisfiability of all concepts. We used the well known Pizza ontology[6] for the $\mathcal{SHON}$ test, in which Pellet required only half the SMT's runtime. After some investigation, we can say that the difference in the second test was mostly caused by Pellet's multiple optimization techniques than by an inherent advantage with modalities.

It is also important to keep in mind that Pellet is an optimized reasoner developed over many years, whereas our implementation is an early stage prototype. In that perspective, we find these results rather promising and they allow us to draw some tentative conclusions. First, the fact that even with the modalities of $\mathcal{ALC}$ hidden by many variables, which prevents the use of many optimizations, the SAT solver computes the problem in a time close to that of a dedicated and highly optimized DL reasoner. Second, the $\mathcal{SHON}$ test has shown that the overhead of the $\mathcal{T}$-solver is acceptable,

[1] `http://www.sat4j.org`
[2] `http://www.cs.sfu.ca/research/groups/mxp/mxc`
[3] `http://www.cs.ubc.ca/labs/projects/satzilla`
[4] `http://clarkparsia.com/pellet`
[5] `http://www.cs.manchester.ac.uk/substance.owl`
[6] `http://www.co-ode.org/ontologies/pizza/pizza.owl`

considering it is not optimized in any way. On that aspect, we are confident that some efficient optimizations from tableau reasoning can be adapted to improve our $\mathcal{T}$-solver's performance.

6 Conclusion

SMT-based systems are usually targeted at relatively simple theories and perform very well in that context. The preliminary results we have at this point allow us to believe that SMT-based approaches can do as well with complex theories since these theories also have a large propositional component.

Although our reasoner is not as good as Pellet, the experimental results we obtained are still good enough to motivate further investigations regarding parallelization and/or distribution of reasoning using this approach, and thus leverage advances in SAT solving in that domain. Investigating in these directions is our next research objective.

References

[BCM+03] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[BS96] M. Böhm and E. Speckenmeyer. A fast parallel sat-solver - efficient workload balancing. In *Annals of Mathematics and Artificial Intelligence*, pages 40–0, 1996.

[GS96] F. Giunchiglia and R. Sebastiani. A SAT-based decision procedure for $\mathcal{ALC}$. In *Proc. of the 5th Intl Conf. on Principles of KR and Reasoning(KR'96)*, pages 304–314. Morgan Kaufmann, 1996.

[KH08a] U. Keller and Stijn Heymans. The sat-tableau calculus. In *Proc. of the 2008 International Workshop on Description Logics (DL2008), Dresden, Germany, May 13 - 16, 2008*, CEUR Workshop Proceedings, 2008.

[KH08b] U. Keller and Stijn Heymans. The sat-tableau calculus: Integrating sat solvers into description logic reasoning. Technical report, Semantic Technology Institute (STI), University of Innsbruck, 2008.

[Sch91] K. Schild. A correspondence theory for terminological logics: Preliminary report. In *In Proc. of IJCAI-91*, 1991.

[SSS91] M. Schmidt-Schaubßand G. Smolka. Attributive concept descriptions with complements. *Artif. Intell.*, 48(1):1–26, 1991.

[SV06] R. Sebastiani and M. Vescovi. Encoding the satisfiability of modal and description logics into sat: The case study of k(m)/alc. In *Proc. of the 2006 International Workshop on Description Logics (DL2006)*, CEUR Workshop Proceedings, pages 130–135. CEUR-WS.org, 2006.

[ZBH96] H. Zhang, M. Bonacina, and J. Hsiang. Psato: a distributed propositional prover and its application to quasigroup problems. *Journal of Symbolic Computation*, 21:543–560, 1996.