The SAT-Tableau Calculus

Uwe Keller[1] and Stijn Heymans[2,3]

[1] Semantic Technology Institute (STI) Innsbruck, University of Innsbruck,E-mail: uwe.keller@sti-innsbruck.at
[2] Knowledge Based Systems Group, Institute of Information Systems, Technical University of Vienna, E-mail: heymans@kr.tuwien.ac.at
[3] Computational Web Intelligence, Department of Applied Mathematics and Computer Science, Ghent University, E-mail: Stijn.Heymans@UGent.be

1 Introduction

Recently, [7] pointed out that the increasing use of Description Logics (DLs) in areas such as e-Science and the Semantic Web is already stretching the capabilities of existing DL systems, posing a range of challenges for future research on reasoning methods for DL. A key problem is the provision of efficient algorithms that allow (advanced) applications (i) to *scale up* to knowledge bases of practical relevance and (ii) to *leverage expressive languages* for capturing domain knowledge. However, expressiveness of DLs comes at a price: the theoretically high (worst-case) complexity of relevant reasoning tasks. Hence, it is very unlikely that there exists a *single method* that performs well in *all* possible cases. Rather, one can expect that specific techniques perform well on particular classes of problems. This motivates the investigation of novel approaches to DL reasoning.

The SAT problem for Propositional Logics on the other hand is a core problem in artificial intelligence. A wealth of algorithms [6] and effective optimization techniques (e.g. [10]) have been designed. Today, the community reached a state where SAT solvers can tackle large-scale problems in industrial applications. A substantial empirical and theoretical body of knowledge on the classes of problems on which certain algorithms perform specifically well has been acquired over the years. Moreover, research on the propositional SAT problem is still a very active field of research. Novel algorithms are still being introduced as it has been demonstrated recently e.g. with the invention of Survey Propagation [1].

It seems therefore natural to investigate how to integrate and exploit state-of-the-art SAT solvers in DL reasoning. In this paper, we present and discuss a particular, rather natural approach to achieve such an integration: the *SAT-Tableau Calculus* is a novel calculus for Description Logic reasoning. It achieves the integration of a wide range of known SAT solvers into *tableau-based* DL reasoning.

Our calculus can be understood as a SAT-Modulo-Theories (SMT) approach [11] that is based on a specific theory-reasoning component. The theory reasoner maintains a finite representation of a (possibly infinite) Kripke-structure and guides the generation of this structure by systematic interaction with a SAT solver. The evolution of the structure is realized by (fairly common) tableau-based completion rules for DLs. The approach therefore combines the power of modern SAT solving techniques with the most

flexible and successful inference technique for DL reasoning, namely *tableau-based* procedures. In contrast to standard SMT-based methods, our approach is not bound to a specific SAT solver but allows to change the SAT solver during the DL reasoning process on demand. Hence, the resulting integration is modular, loose and extremely flexible.

We are not the first ones to investigate the use of SAT solvers for DL reasoning. Perhaps the earliest explicit approach in this direction is KSAT [5]. KSAT had a decisive influence in the development of modern optimized tableau-based systems, e.g. [15]. KSAT is certainly the approach that is closest related to SAT-Tableau and would nowadays be characterized as a SMT-based procedure. In fact, it can be derived from the SAT-Tableau Calculus as presented in the following as a special case [9]. Recently, there have been further approaches [12,14] that use SAT solvers for DL reasoning in specific ways different from the SAT-Tableau Calculus. Both showed promising results when being tested on rather simple logics. For a more detailed discussion, we refer to [9].

Our solution improves upon these approaches in various significant ways:

- **Type of SAT solver that can be integrated:** the SAT-Tableau Calculus takes an abstract view on SAT solving algorithms and treats SAT solvers as black-boxes. In principle, any algorithm that is able to generate propositional models can be integrated within the SAT-Tableau Calculus. Prominent examples are DPLL or Clause Learning procedures, OBDDs, Stochastic Local Search (SLS) Methods, Randomized Algorithms, Resolution- and Analytic Tableau variants, or even very new SAT methods such as B-Cubing or Survey Propagation. By design of the calculus, various SAT solvers can be used simultaneously and changed during runtime as needed.
- **Expressiveness of DLs that can be handled:** the approach cleanly separates two levels of reasoning: the purely propositional level and the purely modal level. The latter is dealt with by a tableau construction procedure. Therefore, the method naturally works for any DL for which a standard tableau-based inference system can be designed or is already known. This especially includes expressive DLs that lack the finite model or the tree model property (e.g. $\mathcal{SHOIN}$ or $\mathcal{SROIQ}$).
- **Reasoning task that can be supported:** we present our calculus here for testing the satisfiability of a concept. We further allow to consider background knowledge in form of a terminology (or TBox). The latter is not covered by any of the approaches mentioned above. Being a tableau-method, the SAT-Tableau Calculus can naturally be used for a more generic problem to which a variety of interesting reasoning tasks for DLs can be reduced efficiently, i.e. checking *ABox consistency*. Hence, the SAT-Tableau Calculus can be extended to ABox reasoning, whereas this seems not possible or unclear for the above mentioned approaches.
- **The way heuristic search can be performed in implementations:** calls to a SAT solver in our approach can be seen as (atomic) macro-inference steps. Our presentation of the calculus takes an abstract view on a propositional SAT solver as an entity that returns propositional models for an input problem. We do not require to only deliver a single model, but in general an (arbitrary) set of models is returned. Since these propositional models are the (atomic) elements to determine (potentially costly and failing) extensions to the Kripke-structure maintained in the

theory reasoning component, our calculus can take an *informed (heuristic) decision* what models to choose to guide the tableau-construction process . This is not the case in current tableau-based systems. Therefore, our calculus naturally suggests a possibility for forward search heuristics at a conceptually suitable level.

By considering instantiations of the SAT-Tableau Calculus with a specific SAT solver, we cannot only derive decision procedures that are well-known [9] (and are amongst the most successful procedures for DL reasoning so far), but we get immediately a range of novel decision procedures that have not been thought of yet, such as BDD-Tableau or SLS-Tableau. Combining two well-investigated areas of research allows further to exploit implementation and optimization techniques from both domains. Hence, we expect that the SAT-Tableau-derived procedures can be at least competitive with state-of-the-art systems, but have a lot of potential to lead us beyond what is currently possible.

Given the flexibility of integrating manifold types of SAT solvers, our tableau calculus inherently is able to adapt better to specific DL problem sets. Since there are various algorithms that work well on different types of SAT problems, we gain flexibility to adapt to the specific kind of SAT problems encountered in a particular application. For these reasons we consider our approach as specifically appealing in regard of the above mentioned research challenges (i) and (ii) identified in [7].

In this paper, we consider the DL $\mathcal{ALC}$ and the problem of checking concept satisfiability wrt. general terminologies. We skip formal definitions as they are common knowledge within the DL community [3]. For further details and formal proofs, we refer the interested reader to an extended technical report [9].

2 The SAT-Tableau Calculus

We introduce the SAT-tableau calculus as an abstract non-deterministic method for deciding concept satisfiability wrt. a terminology $\mathcal{T}$. The abstract calculus can be refined into various deterministic inference procedures by setting a specific deterministic procedure for propositional reasoning and a deterministic strategy for applying the individual inference rules. In particular, standard tableau calculi for DLs are concrete instances of the SAT-tableau calculus which use a specific propositional deduction strategy. Correctness, completeness, and termination can be discussed on the level of the abstract procedure *without* considering specific details of the propositional deduction method but merely focussing on abstract properties of propositional SAT methods. Completeness, correctness, and termination of *any concrete instance* can then be established very easily by verifying a refinement condition demonstrating that the SAT procedure has the required abstract properties.

2.1 A Propositional Logic Perspective on Description Logics

We start by defining the most elementary concepts and notions that are needed to describe our calculus. These notions in principle reflect the informal idea of separating

propositional reasoning and modal reasoning in a two-phased process as motivated informally in Section 3 of [9]. Our presentation here is inspired by [13], a beautiful, informal introduction to SMT in general by Roberto Sebastiani. We specialize this generic viewpoint to the case of DLs and give formal proofs for our setting.

Definition 1 (Concept Subexpressions, Concept Subexpression Literal). *Let Σ be a signature and $C \in \mathcal{C}(\Sigma)$, i.e., a concept expression over the signature Σ. A **concept subexpression of** C is a subexpression C' of C such that $C' \in \mathcal{C}(\Sigma)$. We denote the set of all concept subexpressions of C by $sub(C)$, i.e. the smallest set of concept expressions that contains C and is closed under concept subexpressions. We call a concept subexpression C' of C a **strict concept subexpression** iff. $C' \neq C$. A **concept subexpression literal** is a concept subexpression $C \in sub(C)$ or its complement $\neg C$. We denote by $sublit(C)$ the set of concept subexpression literals.*

Note that, for any concept C the size of $sub(C)$ is linearly bounded in the size of C. The same holds for $sublit(C)$.

Definition 2 (Propositional Atom, Propositional Literal). *Let Σ be a signature and $C \in \mathcal{C}(\Sigma)$. A **propositional atom of** C is a concept subexpression C' of C that is not of the form $D \sqcap D'$, $D \sqcup D'$, or $\neg D$ and is not itself a strict concept subexpression of any propositional atom of C. A **propositional literal of** C is a propositional atom C' of C or its negation $\neg C'$. We denote the set of all propositional atoms of C by $patoms(C)$ and the set of all propositional literals by $pliterals(C)$.*

Essentially, the definition specifies a propositional atom of C as being any longest (or in regard of the syntax tree top-most) non-propositionally decomposable concept subexpression of C. Clearly, for any concept C it holds that $patoms(C) \subseteq sub(C)$ and therefore the size of $patoms(C)$ and $pliterals(C)$ is linearly bounded in the size of C.

Definition 3 (Propositional Interpretation, Partial Propositional Truth Assignment). *Let Σ be a signature and $C \in \mathcal{C}(\Sigma)$. A **propositional interpretation for** C is a set μ of propositional literals of C such that for any propositional atom A of C either $A \in \mu$ or $\neg A \in \mu$ (and not both). A **partial propositional truth assignment for** C is a subset of a propositional interpretation μ of C. For two partial propositional truth assignments μ_1, μ_2 we say μ_2 **extends** μ_1 if $\mu_1 \subseteq \mu_2$.*

The following definition provides the basis for a propositional view on concept expressions:

Definition 4 (Propositional Satisfaction, Entailment). *Let Σ be a signature and $C \in \mathcal{C}(\Sigma)$. A propositional interpretation μ **satisfies** C **propositionally**, denoted by $\mu \models_0 C$, inductively as follows: for any $A \in patoms(C)$, $\mu \models_0 A$ iff. $A \in \mu$; $\mu \models_0 \neg C$ iff. $\mu \not\models_0 C$; $\mu \models_0 C \sqcap D$ iff. $\mu \models_0 C$ and $\mu \models_0 D$; $\mu \models_0 C \sqcup D$ iff. $\mu \models_0 C$ or $\mu \models_0 D$. A partial propositional truth assignment π **satisfies** C **propositionally** (or is a **propositional model** of C) iff. all propositional interpretations μ extending π satisfy C propositionally. Propositional satisfiability is extended to sets $\mathcal{C}$ of concepts as usual: $\pi \models_0 \mathcal{C}$ iff. $\pi \models_0 C$ for all $C \in \mathcal{C}$. A concept expression C **propositionally entails** a concept expression D (denoted by $C \models_0 D$) iff. every propositional model of C is a propositional model of D.*

The notion of propositional satisfaction of concept expressions is identical to the notion of satisfaction in Propositional Logics, i.e. if we consider concept expressions C as propositional formulae over the signature $patoms(C)$, where propositional atoms are not being interpreted in detail, but only considered as atomic propositional symbols. We defined the notion of satisfaction wrt. *partial* propositional interpretations since often it is not necessary to assign a truth value to all propositional atoms to determine the truth value of a concept under this interpretation. In fact, many SAT solvers (which are used later) return only partial propositional interpretations.

Definition 5. *Let Σ be a signature, $C \in \mathcal{C}(\Sigma)$ and $\mathcal{M} = \{\pi_1, \ldots, \pi_k\}$ be a set of partial propositional truth assignments for C. We say that $\mathcal{M}$ **is sound for** C iff. $\pi_i \models_0 C$ for all $\pi_i \in \mathcal{M}$. We say that $\mathcal{M}$ **is complete for** C iff. for any propositional interpretation μ of C such that $\mu \models_0 C$ there is a $\pi_i \in \mathcal{M}$ such that $\pi_i \subseteq \mu$.*

Sound and complete sets of partial propositional truth assignments can be seen as compact representations of the propositional semantics (or models) of C.

We can show the following corollary which provides the basis for our approach:

Corollary 1. *Let Σ be a signature, $C \in \mathcal{C}(\Sigma)$ and $\mathcal{M} = \{\pi_1, \ldots, \pi_k\}$ be a sound and complete set of partial propositional truth assignments for C. Then, C is satisfiable iff. π_i is satisfiable (under the DL semantics) for some $\pi_i \in \mathcal{M}$.*

In other words, the satisfiability of a concept C can be determined by an *iterated* two-phase process: In the first phase we put on our "Propositional Logic glasses" which abstract away from all non-propositional details of the input problem and perform boolean reasoning only, i.e. we (non-deterministically) guess a partial propositional truth assignment $\pi_i = \{l_1, \ldots, l_k\}$ which satisfies C propositionally. In a second step, we put on our "Description Logic glasses" and consider the resulting propositional model under DL semantics. We test if this way we can indeed derive a model. We therefore check the joint satisfiability of all propositional literals $l_j \in \pi_i$ under DL semantics.

The set of concept expressions $\{l_1, \ldots, l_k\}$ that need to be considered during the second phase are in general *simpler* than the concept C: it consists of propositional literals only and therefore contains no further propositional structure (e.g. concept disjunctions) at the top-level. This means, that we are left with *purely modal statements*, which request the existence of certain neighbors (of our current context or individual) with specific properties. We can therefore solve the joint satisfiability of these modal statements by explicitly constructing a graph of nodes which represent individuals in the domain. The nodes are labeled with a set of concept expression which characterize logical properties that must be satisfied by the respective individuals. Given an individual under consideration and a propositional interpretation capturing the elementary logical properties we assume the individual to have, we can extend the graph structure straightforwardly according to the semantics of the modal concept constructors and the required truth value for the propositional literal.

Such a graph structure is usually called *tableau* and technically underlies any tableau-based decision procedure for DLs. We therefore describe a tableau-based approach and

call our method *SAT-Tableau Calculus*. Further, the approach allows to apply all known methods for ensuring the termination of the tableau-construction process, i.e. so-called *blocking techniques* for expressive DLs or reasoning tasks that include terminologies.

Indeed, any individual created during the graph construction becomes an additional DL satisfiability problem in itself: the individual is required to have certain logical properties (specified as DL concept expressions) and we need to find a interpretation in which the individual in fact satisfies these properties. We therefore can solve the created satisfiability subproblems with the very same approach and *iteratively* apply our method to the various nodes that we create in the graph structures. The explicit representation of the tableau is used during the construction process to ensure that the single individual views selected locally by the SAT solvers (by looking at local properties only), are eventually globally consistent with each other.

This means we combine two orthogonal algorithms to form a general purpose DL satisfiability test: (a) a propositional SAT procedure that can enumerate propositional models of a (set of) concept(s) and (b) a tableau-construction process to ensure that the selected propositional views on the individuals in the domain are globally consistent with each other. In general, the algorithm for (b) differs amongst the various DLs (or modal logics) and is the main point of variation.

Please note, that all proofs only use the propositional concept constructors and the notion of a propositional atoms. Hence, all previous propositions, lemmas and corollaries in fact (after a suitable adaption of the notion of a propositional atom) carry over to *any* DL in which the propositional constructors are interpreted classically.

2.2 SAT-Tableau, Completion Graphs, Completion Rule System

Satisfiability testing is essentially a search problem: given a concept (and a terminology) find an interpretation $\mathcal{I}$ such that the input concept describes a property that holds for at least one individual in $\mathcal{I}$. Hence, a natural first step is to derive the search space for our procedure and representation of the search states.

Conceptually, we need to search all (relevant) interpretations over a given signature. Hence, our search space consists of all (relevant) Σ-interpretations. We can consider an interpretation $\mathcal{I}$ over a signature Σ as a graph structure, where each node represents an individual, arcs denote inter-relations of individuals in $\mathcal{I}$ (that are labeled with the respective role names) and each node is labeled with all the elementary logical properties (i.e. a set of atomic concept names) that the individual possesses.

Definition 6 (SAT-Tableau). *Let Σ be a signature, $\mathcal{T}$ be a terminology over Σ, and $C \in \mathcal{C}(\Sigma)$. Let $\mathbf{R}_{C,\mathcal{T}}$ denote the set of role names occurring in C or $\mathcal{T}$ and 2^S denote the powerset of a set S. A **SAT-tableau for** C **wrt.** $\mathcal{T}$ is a labeled graph $T = (\mathbf{V}, \mathbf{E}, l, s)$ with a non-empty set of vertices $\mathbf{V}$, a function $\mathbf{E} : \mathbf{R}_{C,\mathcal{T}} \rightarrow 2^{\mathbf{V}\times\mathbf{V}}$ assigning role names occurring in C or $\mathcal{T}$ a set of edges between nodes in $\mathbf{V}$, a function $l : \mathbf{V} \rightarrow 2^{sublit(\{C\}\cup\mathcal{T})}$ assigning to each node a set of concept subexpressions in C or $\mathcal{T}$ or their complement, and a function $s : \mathbf{V} \rightarrow 2^{pliterals(sub(\{C\}\cup\mathcal{T}))}$ assigning to each node a propositional interpretation for $l(n)$ such that for all nodes $n \in \mathbf{V}$ it holds that:*

(P1) *$s(n)$ satisfies $l(n)$ propositionally*

(P2) *for any* $\neg\exists R.D \in s(n)$ *and any* $\langle n, n' \rangle \in \mathbf{E}(R)$*:* $\neg D \in l(n')$
(P3) *for any* $\exists R.D \in s(n)$ *there exists an* $\langle n, n' \rangle \in \mathbf{E}(R)$ *such that* $D \in l(n')$
(P4) *there exists a node* $n \in \mathbf{V}$ *such that* $C \in l(n)$
(P5) *for any* $D_1 \sqsubseteq D_2 \in \mathcal{T}$*:* $\neg D_1 \sqcup D_2 \in l(n)$

Intuitively, $l(n)$ marks any individual node n with a set of logical properties that n must have, and $s(n)$ selects a specific propositional view on the logical properties of n.

The single clauses in the definition intuitively capture the following: (P1) ensures that our propositional view $s(n)$ on n is indeed (propositionally) consistent with the required logical properties $l(n)$ of the individual. (P2) and (P3) ensure that the selected propositional view on n is also consistent with the requirement under DL semantics, i.e. $s(n)$ satisfies also all modal requirements. (P4) ensures that there is an individual present in the SAT-Tableau which satisfies the properties captured in the input concept C, hence the input concept is shown to be satisfiable. Finally, (P5) requires that the TBox axioms are satisfied, i.e. each individual satisfies the logical properties required by any TBox axiom.

In essence, a *tableau* is a graph structure that represents an *interpretation* (or Kripke-structure) for the signature that satisfies the input concept *and* a terminology (but might contain redundant information about the logical properties of individuals, which are not explicitly represented in a Kripke-structure.). Hence, tableau define goal states of our model search procedure:

Lemma 1. *Let* Σ *be a signature and* $C \in \mathcal{C}(\Sigma)$ *and* $\mathcal{T}$ *be a terminology over* Σ*. Then,* C *is satisfiable wrt.* $\mathcal{T}$ *iff. there exists a SAT-tableau for* C *wrt.* $\mathcal{T}$*.*

In contrast to interpretations, tableau are purely *syntactic* structures which are accessible to algorithms. Therefore, Lemma 1 suggests a data structure and a procedure determine the satisfiability of concepts wrt. terminologies: to check satisfiability, construct (stepwise) a tableau for the input concept wrt. the given terminology. Unfortunately, tableaus do *not* need to be finite. A second problem is to find the requirement set $l(n)$. What concepts need to go in there? (P2), (P3) and (P4) together suggest that these sets can be found in a iterative process which essentially lets concept expressions flow in the tableau-graph, starting from the node mentioned in (P4).

Hence, our procedure needs to work on *finite representations* of a *partially constructed tableau* for an input concept and a terminology. This data structure is called a *completion graph*. Completion graphs represent the actual data structure that represents search states in our model construction algorithm.

Definition 7 (SAT-Completion Graph). *Let* Σ *be a signature,* $\mathcal{T}$ *be a terminology over* Σ*, and* $C \in \mathcal{C}(\Sigma)$*. Let* $\mathbf{R}_{C,\mathcal{T}}$ *denote the set of role names occurring in* C *or* $\mathcal{T}$ *and* 2^S *denote the powerset of a set* S*. A* ***SAT-completion graph for*** C ***wrt.*** $\mathcal{T}$ *is a labeled graph* $G = (V, E, p, r, \pi, state)$ *with a non-empty but finite set of vertices* V *representing individuals, a (possibly empty) set of edges* $E \subseteq V \times V$ *connecting individuals, a node label function* $p : V \to 2^{sublit(\{C\} \cup \mathcal{T})}$ *assigning to each node a set of properties (as concept subexpressions in* C *or* $\mathcal{T}$*) that are required to hold for the individual, a edge label function* $r : E \to 2^{\mathbf{R}_{C,\mathcal{T}}}$ *assigning roles names to edges in* G*, a function* $\pi : V \to 2^{pliterals(sub(\{C\} \cup \mathcal{T}))}$ *assigning to each node a partial propositional*

*truth assignment for $p(n)$, and a function $state : V \rightarrow \{$*NOTASSIGNED, ASSIGNED, UNSATISFIABLE, REASSIGN*$\}$ capturing the processing status of each node in G. G must contain a node $n \in V$ with $C \in p(n)$.*

*For any nodes $n, n' \in V$ with $\langle n, n' \rangle \in E$ and $r(\langle n, n' \rangle) = R$, we call n' an **R-successor** of n in G. n is called **ancestor** of n' in G if there is a path from n to n' in G whereby the single edges on the path can be labeled arbitrarily.*

*A completion graph G contains a **clash** if there exists a node $n \in V$ such that $state(n) =$*

UNSATISFIABLE*. It is called **clash-free** if it does not contain any clash.*

Essentially, a completion graph is very similar to a SAT-tableau, whereas the variety of properties constraining nodes, edges and interdependencies are not present yet. Therefore, each SAT-tableau corresponds to a completion graph, whereas completion graphs do not necessarily correspond to a SAT-tableau, but allow to represent SAT-tableau partially. Consequently, they are a rather natural data structure for any process that systematically and iteratively tries to construct a SAT-tableau.

Before we can present the abstract SAT-Tableau calculus $\mathcal{T}_{\mathbb{SAT}}$ in detail, we need to introduce two further basic concepts , namely *SAT solvers* and *blocking*.

SAT-Solver. In order to construct a tableau (or completion graph), we need to find propositional models for each individual node in the graph. We achieve this by a dedicated algorithm, a *SAT-Solver*, whose internals are not interesting for our matters here. Hence, we need an abstract model for a propositional solver. The following definition captures our understanding formally:

Definition 8 (Propositional Solver). *Let Σ be a signature. A **propositional solver** $\mathfrak{B}$ is an algorithm which takes as input any set $\mathcal{C} \subseteq \mathcal{C}(\Sigma)$ of concepts and computes partial truth assignments that are sound for $\mathcal{C}$, i.e. $\mathfrak{B}(\mathcal{C}) = \mathcal{M}$ for a set $\mathcal{M} = \{\pi_1, \ldots, \pi_k\}$ of partial truth assignments that are sound for $\mathcal{C}$. A propositional solver $\mathfrak{B}$ is called **complete** iff. $\mathfrak{B}(\mathcal{C})$ is a complete set of partial truth assignments for $\mathcal{C}$. Let $\mathbb{BS}$ denote the set of all propositional solvers.*

In other words, a (complete) propositional solver is an algorithm that is capable of computing and enumerating compact representations of (all) propositional models of the given set of concepts $\mathcal{C}$. An example for a complete boolean solver is the DPLL procedure [4] (*without* the pure literal rule) when exhaustive backtracking over propositional models is performed (i.e the solver is repeatedly called again in the last search state (after returning propositional model) to compute and enumerate steps-by-step all models).

Blocking. A completion graph approximates a SAT-tableau, and eventually needs to represent a SAT-tableau. Since SAT-tableau might be infinite themselves[4], we needs to some how find a way to (finitely) represent infinite pathes in a tableau within a completion graph instead of constructing these pathes explicitly. The idea is simple: we

[4] In $\mathcal{ALC}$ this can only happen if we check concept satisfiability wrt. general terminologies containing recursive definitions.

equip the tableau-construction procedure with some means to detect that there is no need to include a new individual in the completion graph to satisfy a particular modal constraint, but that instead it can *reuse* an individual that is already present. This way, we do not insert individuals unnecessarily.

A standard criterion to reuse an individual (*called ancestor blocking*) is as follows: an individual to be reused is an ancestor of the individual for which a suitable neighbor needs to present and the attached property set contains at least all properties that this individual needs possess:

Definition 9 (Ancestor Blocking). *Let G be a completion graph. A node $n \in G$* ***blocks*** *a node $n' \in G$ iff. n is an ancestor of n' in G and $p(n') \subseteq p(n)$.*

Ancestor blocking is a very simple form of blocking which is sufficient for simple DLs such as $\mathcal{ALC}$. For very expressive DLs (e.g. $\mathcal{SHIQ}$), more sophisticated blocking criteria are needed. Such criteria have already been developed (e.g. *pairwise-ancestor-blocking* [8]) and can be integrated easily in the SAT-Tableau Calculus.

Constructing a SAT-Tableau. Our algorithm for checking concept satisfiability non-deterministically constructs completion graphs for the input concept C by starting with the simplest (or smallest) and in general not fully-expanded completion graph G_0 for C that can be defined as follows:

$$\begin{aligned} G_0 := (V, E, p, r, \pi, state) \text{ with} \\ V = \{n_0\}, E = \emptyset, p = \{n_0 \mapsto \{C\}\}, r = \emptyset, \pi = \emptyset, \\ state = \{n_0 \mapsto \text{NOTASSIGNED}\} \end{aligned} \tag{1}$$

The algorithm then proceeds by iterative (non-deterministic) application of the completion rules described in Fig. 1. Note, that any application of a completion rule from Fig. 1 to a completion graph for C wrt. $\mathcal{T}$ results again in a completion graph for C wrt. $\mathcal{T}$.

Definition 10 (Fully-expanded Completion Graph). *A completion graph G is called* ***fully-expanded*** *iff. none of the rules of the completion rules system in Fig. 1 is applicable anymore.*

In essence, by any application of any of the completion rules in our inference system we convert a completion graph G into another completion graph G' that either satisfies an increasing number of the semantic constraints (P1) - (P5) that identify a SAT-Tableau (and hence our goal state), or mark the completion graph as containing a clash and therefore being a dead-end for our completion process. In a sense, we generate increasingly complete (under)approximations of a SAT-tableau (from which we can immediately read of a model for the input concept and the given terminology).

This way, the algorithm eventually (i.e. in the limit of the construction process) must create a fully-expanded completion graph for C wrt. $\mathcal{T}$. This completion graph either contains a clash (in which case we reached a dead-end and where not successful in finding a model) or it is clash-free. In the latter case, we in fact found a (finite representation) of a tableau and therefore a model.

For a detailed discussion of the rules in the completion rules system presented in Fig. 1, we refer the reader to [9]

Rule	Description
$\rightarrow_{Select}$	if 1. n is a node in G, $state(n) \in \{$NOTASSIGNED, REASSIGN$\}$ 2. $\mathfrak{B} \in \mathbb{S}$ is some propositional solver 3. π_j is some propositional model in $\mathfrak{B}(p(n) \cup \pi(n))$ then set $\pi(n) := \pi_j$ and $state(n) :=$ ASSIGNED
$\rightarrow_{Clash}$	if 1. n is a node in G, $state(n) \in \{$NOTASSIGNED, REASSIGN$\}$ 2. $\mathfrak{B}(p(n) \cup \pi(n)) = \emptyset$ 3. $\mathfrak{B} \in \mathbb{S}$ is some complete propositional solver then set $state(n) :=$ UNSATISFIABLE
$\rightarrow_{\exists}$	if 1. n is node in G, $state(n) \in \{$ASSIGNED, REASSIGN$\}$ 2. $c_i = \exists R.C \in \pi(n)$ and there does not exist an R-successor n' of n in G such that $C \in p(n')$ and 3. there does not exist a node b in G such that b blocks n then create a new R-successor n' of n and set $p(n') := \{C\}$ and set $state(n') =$ NOTASSIGNED and set $\pi(n') := \emptyset$
$\rightarrow_{\forall}$	if 1. n is node in G, $state(n) \in \{$ASSIGNED, REASSIGN$\}$ 2. $c_i = \neg\exists R.C \in \pi(n)$ and n' is an R-successor of n in G such that $\neg C \notin p(n')$ then set $p(n') := p(n') \cup \{\neg C\}$ and set $state(n') =$ REASSIGN
$\rightarrow_{\mathcal{T}}$	if 1. n is node in G, $state(n) \neq$ UNSATISFIABLE 2. $C \sqsubseteq D \in \mathcal{T}$ and $\neg C \sqcup D \notin p(n)$ then set $p(n) := p(n) \cup \{\neg C \sqcup D\}$ and set $state(n) =$ REASSIGN

Fig. 1. Completion Rules of $\mathcal{T}_{\mathbb{SAT}}$ (wrt. a non-empty class $\mathbb{S} \subseteq \mathbb{BS}$ of propositional solvers)

2.3 Properties of the SAT-Tableau Calculus

We can now investigate the formal properties of the SAT-Tableau Calculus $\mathcal{T}_{\mathbb{SAT}}$. It turns out that $\mathcal{T}_{\mathbb{SAT}}$ allows to derive a variety of decision procedures for concept satisfiability wrt. terminologies, if we require that the used set of propositional solvers $\mathbb{S}$ contains at *least one* propositional solver which is complete, e.g. a DPLL procedure or OBDDs.

Lemma 2. *Let Σ be a signature, $\mathcal{T}$ be a terminology over Σ and $C \in \mathcal{C}(\Sigma)$. Then, it holds for $\mathcal{T}_{\mathbb{SAT}}$ that process of exhaustive application of the completion rules from Fig. 1 to the initial completion graph G_0 terminates after finitely many steps and results in a fully expanded completion graph G^* for C wrt. $\mathcal{T}$.*

Lemma 3. *Let Σ be a signature, $\mathcal{T}$ be a terminology over Σ and $C \in \mathcal{C}(\Sigma)$. Further, let $\mathbb{S}$ contain at least one propositional solver $\mathfrak{B}$ that is complete. Then, C has a SAT-tableau wrt. $\mathcal{T}$ if there exist a fully expanded completion graph G^* for C wrt. $\mathcal{T}$ that is derived from G_0 and clash-free.*

Lemma 4. *Let Σ be a signature, $\mathcal{T}$ be a terminology over Σ and $C \in \mathcal{C}(\Sigma)$. Further, let $\mathbb{S}$ contain at least one propositional solver $\mathfrak{B}$ that is complete. Then, there exists a from G_0 derived and fully-expanded completion graph G^* for C wrt. $\mathcal{T}$ that is clash-free, if C has a SAT-tableau wrt. $\mathcal{T}$.*

Soundness, completeness and termination of the SAT-Tableau Calculus (wrt. suitable set $\mathbb{S}$ of propositional solvers) is now an immediate consequence of the previous lemmas:

Definition 11 (Proof). *Let Σ be a signature, $\mathcal{T}$ be a terminology over Σ and $C \in \mathcal{C}(\Sigma)$.*

If there a fully-expanded clash-free completion graph G^* for C wrt. $\mathcal{T}$ which can be derived in $\mathcal{T}_{\mathbb{SAT}}$ from G_0, then we say C **is provable in** *$\mathcal{T}_{\mathbb{SAT}}$* **from** *$\mathcal{T}$. We denote this situation by $\mathcal{T} \vdash_{\mathcal{T}_{\mathbb{SAT}}} C$.*

Theorem 1 (Soundness, Completeness and Termination). *The SAT-Tableau Calculus $\mathcal{T}_{\mathbb{SAT}}$ based on a set $\mathbb{S}$ of propositional solvers containing at least one complete propositional solver $\mathfrak{B}$ is a sound and complete decision procedure for checking satisfiability of concepts $C \in \mathcal{C}(\Sigma)$ wrt. terminologies $\mathcal{T}$, i.e.*

1. *$\mathcal{T} \vdash_{\mathcal{T}_{\mathbb{SAT}}} C$ iff. C is satisfiable wrt. $\mathcal{T}$, and*
2. *$\mathcal{T} \vdash_{\mathcal{T}_{\mathbb{SAT}}} C$ can be decided in finite time*

Runtime Complexity. We can give an upper-bound on the runtime of the presented non-deterministic algorithm for two specific cases: (1) If we consider empty terminologies only, then our method non-deterministically constructs a tree whose depth is linearly bounded in the size of the input concept and whose nodes have out-degrees that are linearly bounded in the size of the input concept. Hence, the algorithm runs in non-deterministic exponential time in the size of the input. (2) If we consider concept satisfiability wrt. terminologies, then the algorithm constructs non-deterministically a tree whose nodes' out-degree is linearly bounded in the size of the input and whose depth is exponentially bounded in the size of the input. Hence, the algorithm runs at most in non-deterministic double exponential time in the size of the input. Not surprisingly, the runtime bounds in both cases are the same as the ones for the standard tableau algorithms presented e.g. in [2]. Given the worst-case complexity of detecting concept satisfiability in $\mathcal{ALC}$, it is clear that there is a lot of potential for optimizations, since the upper-bound for the algorithm runtime exceeds the known worst-case complexity of the reasoning tasks significantly. In principle, this cannot be seen as a weakness of the proposed procedure: in the past it has been demonstrated successfully, that non-deterministic tableau-based calculi can be converted into efficient deterministic decision procedures with acceptable runtime (see e.g. [15]) for many cases. In fact, all major optimization techniques that are known should carry over in our SAT-Tableau framework as presented here rather naturally.

3 Conclusions and Future Work

We presented the *SAT-Tableau Calculus*, a novel calculus for Description Logic (DL) reasoning. It achieves the integration of a wide variety of known SAT solving algorithms into tableau-based DL reasoning. The achieved integration is modular and extremely flexible. Our approach improves on previous work on using (specific) SAT solvers for DL reasoning in various significant ways. Most importantly, we extend the type of SAT

solvers that can be used during the DL reasoning process, the expressiveness of DLs that can be handled, the reasoning tasks that can be supported by the approach and the way heuristic search can be performed in implementations.

Combining two well-investigated areas of research (i.e. SAT/SMT and tableau-based reasoning) further allows to exploit implementation and optimization techniques from both domains. Further, by considering instantiations of the SAT-Tableau Calculus with a specific SAT solver, we can not only derive decision procedures that are well-known [9](and are amongst the most successful procedures for DL reasoning so far), but we get immediately a range of novel decision procedures that have not been thought of yet. Hence, we expect that the SAT-Tableau-derived procedures can be at least competitive with state-of-the-art systems, but have a lot of potential to lead us beyond what is currently possible.

Future Work. An implementation of the SAT-Tableau Calculus for $\mathcal{ALC}$ as presented here is ongoing and first evaluation results for specific instances (i.e. BDD-Tableau and SAT-Tableau based on Stochastic Local Search-based SAT solvers) can be expected soon. Concerning the theory, we will extend the SAT-Tableau calculus in various dimensions: (a) to more expressive DLs (e.g. $\mathcal{SHOIN}$) to investigate potential simplifications of standard tableau systems for these logics, (b) to allow for taking ABoxes and RBoxes into account (and hence general DL knowledge bases), (c) to investigate blocking in a more principle way and, and finally (d) to revisit absorption as an optimization technique with major practical relevance.

Acknowledgments. We would like to thank Enrico Franconi and Peter Patel-Schneider for comments on our ideas. This work has been funded by the Austrian Federal Ministry for Transport, Innovation, and Technology under the project *Semantic Engineering Support Environment* (SEnSE, FIT-IT contract FFG 810807). Stijn Heymans is partially supported by the Austrian Science Fund (FWF) under project P20305-N18 and the Fund for Scientific Research Flanders (FWO Vlaanderen) under project 3G010107.

References

1. R. Zecchina A. Braunstein, M. Mézard. Survey propagation: An algorithm for satisfiability. *Random Structures and Algorithms*, 27(2):201–226, 2005.
2. F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.
3. Franz Baader, Ian Horrocks, and Ulrike Sattler. Description Logics. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*. Elsevier, 2007. To appear.
4. Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.
5. Fausto Giunchiglia and Roberto Sebastiani. A SAT-based Decision Procedure for ALC. In *KR*, pages 304–314, 1996.
6. J. Gu, P. Purdom, J. Franco, and B. Wah. Algorithms for the Satisfiability (SAT) Problem: A Survey, 1996.

7. Ian Horrocks. Applications of description logics: State of the art and research challenges. In Frithjof Dau, Marie-Laure Mugnier, and Gerd Stumme, editors, *Proc. of the 13th Int. Conf. on Conceptual Structures (ICCS'05)*, number 3596 in Lecture Notes in Artificial Intelligence, pages 78–90. Springer, 2005.
8. Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for very expressive description logics. *J. of the Interest Group in Pure and Applied Logic*, 8(3):239–264, 2000.
9. Uwe Keller and Stijn Heymans. The SAT-Tableau Calculus: Integrating SAT Solvers into Description Logic Reasoning. Technical Report STI TR 2008-01-18, Semantic Technology Institute (STI), University of Innsbruck, January 2008. Available for download at: http://www.yodlr-reasoner.net/theory.
10. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *DAC*, pages 530–535, 2001.
11. Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(). *J. ACM*, 53(6):937–977, 2006.
12. Guoqiang Pan, Ulrike Sattler, and Moshe Y. Vardi. Bdd-based decision procedures for the modal logic k. *Journal of Applied Non-Classical Logics*, 16(1-2):169–208, 2006.
13. Roberto Sebastiani. On efficiently integrating boolean and theory-specific solving procedures. Technical Report DIT-04-036, DIT, University of Trento, 2004.
14. Roberto Sebastiani and Michele Vescovi. Encoding the satisfiability of modal and description logics into sat: The case study of k(m)/alc. In *Proceedings of 9th International Conference on the Theory and Applications of Satisfiability Testing*, pages 130–135, 2006.
15. Dmitry Tsarkov, Ian Horrocks, and Peter F. Patel-Schneider. Optimizing terminological reasoning for expressive description logics. *J. of Automated Reasoning*, 39(3):277–316, 2007.