Coupling Tableau Algorithms for Expressive Description Logics with Completion-based Saturation Procedures

Andreas Steigmiller*[1], Birte Glimm[1], and Thorsten Liebig[2]

[1] University of Ulm, Ulm, Germany, <first name>.<last name>@uni-ulm.de
[2] derivo GmbH, Ulm, Germany, liebig@derivo.de

Abstract. Nowadays, saturation-based reasoners for the OWL EL profile are able to handle large ontologies such as SNOMED very efficiently. However, saturation-based reasoning procedures become incomplete if the ontology is extended with axioms that use features of more expressive Description Logics, e.g., disjunctions. Tableau-based procedures, on the other hand, are not limited to a specific OWL profile, but even highly optimised reasoners might not be efficient enough to handle large ontologies such as SNOMED. In this paper, we present an approach for tightly coupling tableau- and saturation-based procedures that we implement in the OWL DL reasoner Konclude. Our detailed evaluation shows that this combination significantly improves the reasoning performance on a wide range of ontologies.

1 Introduction

The current version of the Web Ontology Language (OWL 2) [19] is based on the very expressive Description Logic (DL) $\mathcal{SROIQ}$ [6]. To handle (standard) reasoning tasks, sound and complete tableau algorithms are typically used, which are easily extensible and adaptable. Moreover, the use of a wide range of optimisation techniques allows for handling many expressive, real-world ontologies. Since standard reasoning tasks for $\mathcal{SROIQ}$ have N2EXPTIME-complete worst-case complexity [9], it is, however, not surprising that larger ontologies easily become unpractical for existing systems.

In contrast, the OWL 2 profiles define language fragments of $\mathcal{SROIQ}$ for which reasoning tasks can be realised efficiently, e.g., within polynomial worst-case complexity. For example, the OWL 2 EL profile is based on the DL $\mathcal{EL}^{++}$ which can be handled very efficiently by variants of saturation-based reasoning procedures [2,10]. These saturation algorithms have also been pushed to more expressive DLs (e.g., Horn-$\mathcal{SHIQ}$ [10] or $\mathcal{ALCH}$ [12]) for which they are often able to outperform the more general tableau algorithms. In particular, they allow a fast one-pass handling of several reasoning tasks such as classification (i.e., the task of arranging the named concepts of an ontology in a subsumption hierarchy), whereas tableau-based procedures perform classification by a pairwise comparison of the named concepts. Handling cardinality restrictions with saturation procedures is, however, still an open question.

* The author acknowledges the support of the doctoral scholarship under the Postgraduate Scholarships Act of the Land of Baden-Wuerttemberg (LGFG).

Recently, new approaches have been proposed to also improve the reasoning performance for ontologies of more expressive DLs by combining saturation procedures and fully-fledged tableau reasoners in a black box manner [1,14]. These approaches try to delegate as much work as possible to the specialised and more efficient reasoner, which allows for reducing the workload of the fully-fledged tableau algorithm, and often results in a better pay-as-you-go behaviour than using a tableau reasoner alone.

In this paper, we present a much tighter coupling between saturation- and tableau-based algorithms, whereby further performance improvements are achieved. After introducing some preliminaries (Section 2), we present a saturation procedure that is adapted to the data structures of a tableau algorithm (Section 3). This allows for easily passing information between the saturation and the tableau algorithm within the same reasoning system. Moreover, the saturation partially handles features of more expressive DLs in order to efficiently derive as many consequences as possible (Section 3.1). We then show how parts of the ontology can be identified for which the saturation procedure is possibly incomplete and where it is necessary to fall-back to the tableau procedure (Section 3.2). Subsequently, we present several optimisations that are based on passing information from the saturation to the tableau algorithm (Section 4) and back (Section 5). Finally, we present the results of a detailed evaluation (Section 6) before we conclude (Section 7). Further details, proofs, an extended evaluation, and comparisons with other reasoners are available in a technical report [16].

2 Preliminaries

For brevity, we do not introduce DLs (see, e.g., [3]) and we only present our approach for the DL $\mathcal{ALCHOIQ}$. However, the approach can easily be extended to $\mathcal{SROIQ}$ (see [16]), e.g., by encoding role chains and adding appropriate rules for the remaining features such as $\exists r.\mathsf{Self}$ concepts.

2.1 Tableau Algorithm

For ease of presentation, we assume in the remainder of the paper that all concepts are in negation normal form (NNF) and we use $\dot{\neg}$ to denote the negation of a concept in NNF. Moreover, we assume that all ABox axioms are internalised into the TBox of a knowledge base.[3]

A tableau algorithm decides the consistency of a knowledge base $\mathcal{K}$ by trying to construct an abstraction of a model for $\mathcal{K}$, a so-called "completion graph". A completion graph G is a tuple $(V, E, \mathcal{L}, \dot{\neq})$, where each node $v \in V$ (edge $\langle v, w\rangle \in E$) represents one or more (pairs of) individuals. Each node v (edge $\langle v, w\rangle$) is labelled with a set of concepts (roles), $\mathcal{L}(v)$ ($\mathcal{L}(\langle v, w\rangle)$), which the individuals represented by v ($\langle v, w\rangle$) are instances of. The relation $\dot{\neq}$ records inequalities between nodes.

The algorithm works by initialising the graph with one node for each nominal in the input knowledge base. Complex concepts are then decomposed using a set of expansion rules, where each rule application can add new concepts to node labels and/or

[3] In the presence of nominals, this can easily be realised, e.g., by expressing a concept assertion $C(a)$ (role assertion $r(a, b)$) as $\{a\} \sqsubseteq C$ ($\{a\} \sqsubseteq \exists r.\{b\}$).

new nodes and edges to the completion graph, thereby explicating the structure of a model. The rules are applied until either the graph is fully expanded (no more rules are applicable), in which case the graph can be used to construct a model that is a *witness* to the consistency of $\mathcal{K}$, or an obvious contradiction (called a *clash*) is discovered (e.g., both C and $\dot{\neg}C$ in a node label), proving that the completion graph does not correspond to a model. The input knowledge base $\mathcal{K}$ is *consistent* if the rules (some of which are non-deterministic) can be applied such that they build a fully expanded, clash-free completion graph. A cycle detection technique called *blocking* ensures the termination of the algorithm.

Typically, lazy unfolding rules are used in the tableau algorithm to process axioms of the form $A \sqsubseteq C$, where the concept C is added to the label of a node if it contains the atomic concept A. Axioms that are not directly supported by this lazy unfolding approach must be internalised, which can be realised by expressing a general concept inclusion (GCI) axiom $C \sqsubseteq D$ by $\top \sqsubseteq \dot{\neg}C \sqcup D$. Given that $\top$ is satisfied at each node, the disjunction is then also added to all node labels.

2.2 (Binary) Absorption

Absorption is used as a preprocessing step in order to reduce the non-determinism in the tableau algorithm. Basically, axioms are rewritten into (possibly several) simpler concept inclusion axioms such that lazy unfolding rules in the tableau algorithm can be used and, therefore, internalisation of axioms is often not required. Algorithms based on binary absorption [8] allow for and create axioms of the form $(A_1 \sqcap A_2) \sqsubseteq C$, whereby also more complex axioms can be absorbed. To efficiently support a binary absorption axiom $(A_1 \sqcap A_2) \sqsubseteq C$ in the tableau algorithm, a separate unfolding rule is used, which adds C only to node labels if A_1 and A_2 are already present. More sophisticated absorption algorithms, such as partial absorption [15], further improve the handling of knowledge bases for more expressive DLs since the non-determinism that is caused by disjunctions on the right-hand side of axioms is further reduced. Roughly speaking, the non-absorbable disjuncts are partially used as conditions on the left-hand side of additional inclusion axioms such that the processing of the disjunctions can further be delayed. Many state-of-the-art reasoning systems are at least using some kind of binary absorption, which makes the processing of simple ontologies (e.g., EL ontologies) also with the tableau algorithm deterministic. In the following, we assume that knowledge bases are, at least, preprocessed with a variant of binary absorption and we also use the syntax of binary absorption axioms to illustrate the algorithms and examples.

3 Saturation Compatible with Tableau Algorithms

In this section, we describe a saturation method that is an adaptation of the completion-based procedure [2] such that it generates data structures that are compatible for further usage within a fully-fledged tableau algorithm for more expressive DLs. Similarly to completion graphs, this saturation generates nodes that are labelled with sets of concepts and, therefore, it directly allows for transferring results from the saturation to the tableau algorithm. For example, the saturated labels can be used to initialise the labels of new

nodes in the completion graph or to block the processing of nodes. In some cases, it is directly possible to extract completion graphs from the data structures of the saturation, which makes an explicit model construction with the tableau algorithm unnecessary.

Note, the adapted saturation method is not designed to cover a certain OWL 2 profile or a specific DL language. In contrast, we saturate those parts of a knowledge base that can easily be supported with an efficient algorithm (see Section 3.1). Unsupported concept constructors are (partially) ignored by the saturation, but we dynamically detect which parts have not been completely handled afterwards (see Section 3.2). Hence, the results of the saturation are possibly incomplete, but since we know how and where they are incomplete, we can use the results from the saturation appropriately.

3.1 Saturation based on Tableau Rules

The adapted saturation method generates so-called saturation graphs, which approximate completion graphs in a compressed form (e.g., it allows for "reusing" nodes).

Definition 1 (Saturation Graph). *Let* $\mathsf{Rols}(\mathcal{K})$ *(*$\mathsf{fclos}(\mathcal{K})$*) denote the roles (concepts) that occur in* $\mathcal{K}$ *(in completion graphs for* $\mathcal{K}$*). A* saturation graph *for* $\mathcal{K}$ *is a directed graph* $S = (V, E, \mathcal{L})$ *with the nodes* $V \subseteq \{v_C \mid C \in \mathsf{fclos}(\mathcal{K})\}$*. Each node* $v_C \in V$ *is labelled with a set* $\mathcal{L}(v_C) \subseteq \mathsf{fclos}(\mathcal{K})$ *such that* $\mathcal{L}(v_C) \supseteq \{\top, C\}$*. We call* v_C *the* representative node *for the concept* C*. Each edge* $\langle v, v' \rangle \in E$ *is labelled with a set* $\mathcal{L}(\langle v, v' \rangle) \subseteq \mathsf{Rols}(\mathcal{K})$*. We say that a node* $v \in V$ *is* clashed *if* $\bot \in \mathcal{L}(v)$*.*

A major difference to a completion graph is the missing $\doteq$ relation, which can be omitted since the saturation is not designed to completely handle cardinality restrictions and, therefore, we also do not need to keep track of inequalities between nodes in the saturation graph. Furthermore, each node in the saturation graph is the representative node for a specific concept, which allows for reusing nodes. For example, instead of creating new successors for existential restrictions, we reuse the representative node for the existentially restricted concept as a successor.

In principle, the nodes, edges, and labels are used as in completion graphs (cf. [6]) and, therefore, we also use the (r-)neighbour, (r-)successor, (r-)predecessor, ancestor and descendant relations analogously. Please note, however, that a node in the saturation graph can have several predecessors due to the reuse of nodes.

We initialise the saturation graph with the representative nodes for all concepts that have to be saturated. For example, if the satisfiability of the concept C has to be tested, then we are interested in the saturation of the concept C and, therefore, we add the node v_C with the label $\mathcal{L}(v_C) = \{\top, C\}$ to the saturation graph. Note that we only build one saturation graph, i.e., if we are later also interested in the saturation of a concept D that is not already saturated, then we simply extend the existing saturation graph by v_D. For knowledge bases that contain nominals, we also add a node $v_{\{a\}}$ with $\mathcal{L}(v_{\{a\}}) = \{\top, \{a\}\}$ for each nominal $\{a\}$ occurring in the knowledge base.

For the initialised saturation graph, we apply the saturation rules depicted in Table 1. Note that if a saturation rule refers to the representative node for a concept C and the node v_C does not yet exist, then we assume that the saturation graph is automatically extended by this node. Although the saturation rules are very similar to the

Table 1. Saturation rules for the (partial) handling of $\mathcal{ALCHOIQ}$ knowledge bases

$\sqsubseteq_1$-rule:	if	$H \in \mathcal{L}(v)$, $H \sqsubseteq C \in \mathcal{K}$ with $H = A$, $H = \{a\}$, or $H = \top$, and $C \notin \mathcal{L}(v)$,
	then	$\mathcal{L}(v) \longrightarrow \mathcal{L}(v) \cup \{C\}$
$\sqsubseteq_2$-rule:	if	$\{A, B\} \subseteq \mathcal{L}(v)$, $(A \sqcap B) \sqsubseteq C \in \mathcal{K}$, and $C \notin \mathcal{L}(v)$,
	then	$\mathcal{L}(v) \longrightarrow \mathcal{L}(v) \cup \{C\}$
$\sqcap$-rule:	if	$C_1 \sqcap C_2 \in \mathcal{L}(v)$ and $\{C_1, C_2\} \not\subseteq \mathcal{L}(v)$,
	then	$\mathcal{L}(v) \longrightarrow \mathcal{L}(v) \cup \{C_1, C_2\}$
$\exists$-rule:	if	$\exists r.C \in \mathcal{L}(v)$ and $r \notin \mathcal{L}(\langle v, v_C \rangle)$,
	then	$\mathcal{L}(\langle v, v_C \rangle) \longrightarrow \mathcal{L}(\langle v, v_C \rangle) \cup \{r\}$
$\forall$-rule:	if	$\forall r.C \in \mathcal{L}(v)$, there is an $\mathsf{inv}(r)$-predecessor v' of v, and $C \notin \mathcal{L}(v')$,
	then	$\mathcal{L}(v') \longrightarrow \mathcal{L}(v') \cup \{C\}$
$\sqcup$-rule:	if	$C_1 \sqcup C_2 \in \mathcal{L}(v)$, there is some $D \in \mathcal{L}(v_{C_1}) \cap \mathcal{L}(v_{C_2})$, and $D \notin \mathcal{L}(v)$,
	then	$\mathcal{L}(v) \longrightarrow \mathcal{L}(v) \cup \{D\}$
$\geqslant$rule:	if	$\geqslant n\, r.C \in \mathcal{L}(v)$ with $n \geq 1$ and $r \notin \mathcal{L}(\langle v, v_C \rangle)$,
	then	$\mathcal{L}(\langle v, v_C \rangle) \longrightarrow \mathcal{L}(\langle v, v_C \rangle) \cup \{r\}$
o-rule:	if	$\{a\} \in \mathcal{L}(v)$, there is some $D \notin \mathcal{L}(v)$, and
		$D \in \mathcal{L}(v_{\{a\}})$ or there is a descendant v' of v with $\{\{a\}, D\} \subseteq \mathcal{L}(v')$,
	then	$\mathcal{L}(v) \longrightarrow \mathcal{L}(v) \cup \{D\}$
$\bot$-rule:	if	$\bot \notin \mathcal{L}(v)$, and
		1. $\{C, \neg C\} \subseteq \mathcal{L}(v)$, or
		2. $\{\geqslant n r.C, \leqslant m\, s.D\} \subseteq \mathcal{L}(v)$ with $n > m$, $r \sqsubseteq^* s$ and $D \in \mathcal{L}(v_C)$, or
		3. $\geqslant n r.C \in \mathcal{L}(v)$ with $n > 1$, and $\{a\} \in \mathcal{L}(v_C)$, or
		4. there exist a successor node v' of v with $\bot \in \mathcal{L}(v')$, or
		5. there exist a node $v_{\{a\}}$ with $\bot \in \mathcal{L}(v_{\{a\}})$,
	then	$\mathcal{L}(v) \longrightarrow \mathcal{L}(v) \cup \{\bot\}$

corresponding expansion rules in the tableau algorithm, there are some differences. For example, the number of nodes is limited by the number of (sub-)concepts occurring in the knowledge base due to the reuse of nodes for satisfying existentially restricted concepts. Consequently, the saturation is terminating since the rules are only applied when they can add new concepts or roles to node or edge labels. Moreover, a cycle detection technique such as blocking is not required, which makes the rule application very fast. Note also that the $\forall$-rule propagates concepts only to the predecessors of a node, which is necessary in order to allow the reuse of nodes for existentially restricted concepts. Language features of more expressive DLs are only partially supported. For instance, the $\sqcup$-rule adds only those concepts that are implied by both disjuncts. In order to (partially) handle a nominal $\{a\}$ in the label of a node v, we use an o-rule that adds those concepts that are derived for $v_{\{a\}}$ or for descendant nodes that also have $\{a\}$ in their label (instead of merging such nodes as in tableau procedures). This enables a very efficient implementation and is sufficient for many ontologies.

The rules also include a $\bot$-rule, which adds the concept $\bot$ to the label of those nodes for which a clash can be discovered. Furthermore, it propagates $\bot$ to the ancestor nodes. In case $\bot$ occurs in the label of a representative node for a nominal, the knowledge base is inconsistent and $\bot$ is propagated to every node label in the saturation graph; otherwise $\bot$ in the label of a node v_C indicates the unsatisfiability of C. Although it is, in principle, possible to detect also several other kinds of clashes for the incompletely handled parts

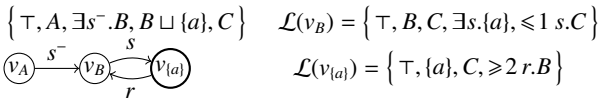

Fig. 1. Generated saturation graph for testing the satisfiability of A_1 for Example 1

in the saturation (e.g., for a concept C that has to be propagated to a successor node v, where v has already the negation of C in its the label), the presented conditions of the $\bot$-rule are already sufficient to show the completeness. Hence, we omit further clash conditions for ease of presentation. Note, the use of a $\bot$-rule is typical for saturation procedures since we are interested in associating clashes with specific nodes instead of entire completion graphs. As a consequence, the saturation allows for handling several independent concepts within the same saturation graph, while unsatisfiable nodes can nevertheless be distinguished from nodes that are (possibly) still satisfiable.

Example 1. Let us assume that the TBox $\mathcal{T}$ contains the following axioms:

$$A \sqsubseteq \exists s^-.B \qquad A \sqsubseteq B \sqcup \{a\} \qquad B \sqsubseteq C \qquad B \sqsubseteq \exists s.\{a\}$$
$$B \sqsubseteq \leqslant 1\ s.C \qquad \{a\} \sqsubseteq C \qquad \{a\} \sqsubseteq \geqslant 2\ r.B$$

In order to test the satisfiability of the concept A, we initialise the saturation graph with the representative node for A and the nominal $\{a\}$. Applying the rules of Table 1 yields the saturation graph depicted in Figure 1. Note that the procedure creates new nodes on demand, e.g., for the processing of disjunctions, existential restrictions, and at-least cardinality restrictions. Although the concept C is added to node labels, a node for C is not created since C is not used in a way that requires this. Also note that the $\sqcup$-rule application adds C to the label of v_A, because C is in the label of the representative nodes for both disjuncts of the disjunction $B \sqcup \{a\}$ (i.e., $C \in \mathcal{L}(v_B) \cap \mathcal{L}(v_{\{a\}})$).

With a suitable absorption technique, the saturation is usually able to derive and add the majority of those concepts that would also be added by the tableau algorithm for an equivalent node. This is especially the case for ontologies that primarily use features of the DL $\mathcal{EL}^{++}$. Since $\mathcal{EL}^{++}$ covers many important and often used constructors (e.g., $\sqcap$, $\exists$), the saturation does already the majority of the work for many ontologies (as confirmed by our evaluation in Section 6).

3.2 Saturation Status Detection

If used alone, the presented saturation procedure easily becomes incomplete for more expressive DLs, similarly to other saturation-based procedures. Our aim is, however, to gain as much information as possible from the saturation, i.e., we would like to detect more precisely for which nodes the saturation was incomplete. In principle, this can easily be approximated by testing for which nodes the actual tableau expansion rules are applicable. However, since we partially saturate some more expressive concept constructors, this approach is often too conservative. For example, consider a saturation graph without nominals and at-most cardinality restrictions, but with an at-least cardinality restriction $\geqslant n\ r.C$ with $n > 1$ in some node label. When constructing a model

from the saturation graph, we could create the required n successors by "copying" the node v_C. Nevertheless, the tableau expansion rule for this at-least cardinality restriction is still applicable since we only have one successor. It would, however, be sufficient to check whether the number of successors is possibly limited by at-most cardinality restrictions or nominals. Similar relaxations are also possible for other concept constructors, which is exploited by the approach described in this section.

In order to identify nodes for which the saturation procedure might be incomplete, we first identify nodes that depend (directly or indirectly) on nominals and nodes that have tight at-most restrictions.

Definition 2. *Let $S = (V, E, \mathcal{L})$ be a saturation graph and $v \in V$ a node. We say that v is* directly nominal dependent *if $\{a\} \in \mathcal{L}(v)$; v is* nominal dependent *if v is directly nominal dependent or v has a successor node v' such that v' is nominal dependent.*

For a role s and a concept D, the number of merging candidates for v w.r.t. s and D *is defined as $\sum_{\geqslant n\, r.C \in G} n$ with*

$$G = \{\geqslant n\, r.C \in \mathcal{L}(v) \mid r \sqsubseteq^* s \text{ and } D \in \mathcal{L}(v_C)\} \cup \{\geqslant 1\, r.C \mid \exists r.C \in \mathcal{L}(v), r \sqsubseteq^* s \text{ and } D \in \mathcal{L}(v_C)\}.$$

The node v has tight at-most restrictions *if there is an at-most cardinality restriction $\leqslant m\, s.D \in \mathcal{L}(v)$ and the number of merging candidates for v w.r.t. s and D is exactly m.*

For nodes with tight at-most restrictions, it is not necessary to merge some of its merging candidates, but every additional candidate might require merging and, therefore, these nodes cannot be used arbitrarily.

We can now identify critical nodes that are possibly incompletely handled by the saturation as follows:

Definition 3. *Let $S = (V, E, \mathcal{L})$ be a saturation graph and $v \in V$ a node. We say that v is* directly critical*, if*

- C1 $\forall r.C \in \mathcal{L}(v)$ *and there is an r-successor v' of v such that $C \notin \mathcal{L}(v')$;*
- C2 $C \sqcup D \in \mathcal{L}(v)$ *and $C, D \notin \mathcal{L}(v)$;*
- C3 $\leqslant m\, s.D \in \mathcal{L}(v)$ *and there is an s-successor v' of v such that $\mathcal{L}(v') \cap \{D, \dot{\neg} D\} = \emptyset$;*
- C4 $\leqslant m\, s.D \in \mathcal{L}(v)$ *and the number of merging candidates for v w.r.t. s and D is greater than m;*
- C5 *v has an* $\mathsf{inv}(s)$*-successor v' with $\leqslant m\, s.D \in \mathcal{L}(v')$ and $\mathcal{L}(v) \cap \{D, \dot{\neg} D\} = \emptyset$;*
- C6 *v has an* $\mathsf{inv}(s)$*-successor v' with $\leqslant m\, s.D \in \mathcal{L}(v')$, $D \in \mathcal{L}(v)$, and the number of merging candidates for v' w.r.t. s and D is m;*
- C7 $\{a\} \in \mathcal{L}(v)$ *and there is some $v' \in V$ with $\{a\} \in \mathcal{L}(v')$ and $\mathcal{L}(v) \not\subseteq \mathcal{L}(v')$;*
- C8 *v is nominal dependent and for some nominal $\{a\}$ the node $v_{\{a\}}$ is critical; or*
- C9 *v has an* $\mathsf{inv}(s)$*-successor v' with $\leqslant m\, s.D \in \mathcal{L}(v')$ and $\{a\} \in \mathcal{L}(v')$.*

We say that v is critical *if v is directly critical or v has a critical successor v'.*

Conditions C1, C2, and C3 identify nodes as critical for which the $\forall$-, the $\sqcup$-, or the ch-rule of the tableau algorithm is applicable. Note that in Condition C1 it is only necessary to check whether the concept can be propagated to successor nodes since the propagation to predecessors is ensured by the saturation procedure. Condition C4 identifies nodes as critical for which at-most restrictions might not be satisfied. Conditions C5 and C6 work analogously to C3 and C4, but check this from the perspective

of a predecessor node. Note that C6 only has to check whether the number of merging candidates is equal to m since nodes with an at-most cardinality restriction $\leqslant m\, s.D$ and more merging candidates than m are already critical due to C4. Condition C7 checks whether merging different nodes in the saturation graph that have the same nominal in their label could lead to problems, while C8 marks nominal dependent nodes as critical if representative nodes for nominals are critical since it cannot be excluded that more consequences are propagated to these nodes over the nominals. Finally, Condition C9 identifies nodes as critical for which an interaction between at-most restrictions, nominals and inverse roles could occur and thus the NN-rule of the tableau algorithm could be applicable.

A concept C is obviously unsatisfiable if its representative node is clashed (i.e., $\bot \in \mathcal{L}(v_C)$), whereas the satisfiability of C can only be guaranteed (for the general case) if v_C is not critical, v_C does not depend on a nominal, and the knowledge base is consistent. Consistency is explicitly required, because a concept is satisfiable only if the knowledge base is consistent, which, however, cannot always be determined by the saturation procedure since it might not be able to completely handle all representative nodes for nominals. In particular, if the saturation graph contains a critical representative node for a nominal, then only the nominal dependent nodes are also marked as critical. Thus, for the remaining nodes, we have to require that the knowledge base is consistent in order to be able to guarantee the satisfiability of their associated concepts. In addition, if a node v_C is nominal dependent, then the consequences that are propagated to v_C obviously depend on the labels of the corresponding representative nodes for these nominals. Therefore, we cannot generally guarantee the satisfiability of C without knowing the status of the representative nodes for those nominals on which v_C depends.

Please also note that a critical representative node for a nominal also makes all nominal dependent nodes critical, which can obviously be very problematic in practice. In Section 5, we show how we can use information from a completion graph, e.g., from the initial consistency check, to improve the status of the saturation graph.

Example 1 (continued). For the saturation graph depicted in Figure 1, v_A, v_B, and $v_{\{a\}}$ are nominal dependent, v_B has a tight at-most restriction, and only v_A is critical: First, C6 applies to v_A since v_B is an s^--successor of v_A due to $\exists s^-.B \in \mathcal{L}(v_A), \leqslant 1\, s.C \in \mathcal{L}(v_B)$ and the number of merging candidates for v_B w.r.t. s and C is 1. Second, C2 applies to v_A since none of the disjuncts of $B \sqcup \{a\} \in \mathcal{L}(v_A)$ occurs in $\mathcal{L}(v_A)$.

4 Assisting Tableau Algorithms

In this section, we show how we can use the saturation graph to improve the tableau algorithm such that existing optimisations can still be used. For example, to further support the important dependency directed backtracking [3,18], which allows for evaluating only relevant non-deterministic alternatives, we have to correctly manage the dependencies for all results that we transfer from the saturation into a completion graph.

4.1 Transfer of Saturation Results to Completion Graphs

Since the saturation uses compatible data structures, we can directly transfer the saturation results into the completion graph. For example, if we create a new successor node v

due to an existential restriction $\exists r.C$, then we can directly initialise v with the concepts from $\mathcal{L}(v_C)$ and record that the added concepts deterministically depend on C. The most notable advantage of the transferred consequences is that they often allow for blocking much earlier. Basically, concepts that would be propagated back from successor nodes are already present in the node label and, thus, a block can often be established even without creating and processing the required successors.

Furthermore, the successors of a node v in the completion graph can be blocked if there is a node v' in the saturation graph such that v and v' are labelled with the same concepts and v' is neither clashed, critical nor nominal dependent. If v' is nominal dependent and we would block the successors of v, then we might miss the handling of new consequences if the dependent nominal nodes are modified in this completion graph. If v' does not have a tight at-most restriction, then we can directly block v since merging with a predecessor can be excluded. Of course, if new concepts are propagated to v, then the block becomes invalid and the processing of the successors has to be reactivated unless another node can be used for the blocking.

4.2 Subsumer Extraction

Higher level reasoning tasks such as classification often exploit information that can be extracted from the constructed completion graphs [5]. Obviously, we can also use the saturation graph to improve classification. For example, if a node v_A is neither clashed nor critical, then A is satisfiable and $\mathcal{L}(v_A)$ contains all of its subsumers. In particular, if no nodes are critical (which is the case for many EL ontologies), only a transitive reduction is necessary for classification and, thus, we automatically get a one-pass classification for simple ontologies. Otherwise, the subsumers identified by the saturation can be used to initialise the tableau-based classification algorithm, which is more accurate than the often used told subsumers extracted from the ontology axioms.

4.3 Model Merging

Many ontologies contain axioms of the form $C \equiv D$, which can be seen as an abbreviation for $C \sqsubseteq D$ and $D \sqsubseteq C$. Treating axioms of the form $A \equiv D$ with A an atomic concept as $A \sqsubseteq D$ and $D \sqsubseteq A$ can, however, downgrade the performance of tableau algorithms since absorption might not apply to $D \sqsubseteq A$, i.e., the axiom is internalised into $\top \sqsubseteq \dot{\neg} D \sqcup A$. To avoid this, many implemented tableau algorithms explicitly support $A \equiv D$ axioms by an additional unfolding rule, where the concept A in the label of a node is unfolded to D and $\neg A$ to $\dot{\neg} D$ (exploiting that $D \sqsubseteq A$ is equivalent to $\neg A \sqsubseteq \dot{\neg} D$) [7].[4] Unfortunately, using such an unfolding rule also comes at a price since the tableau algorithm is no longer forced to add either A or $\dot{\neg} D$ to each node in the completion graph, i.e., we might not know for some nodes whether they represent instances of A or $\neg A$. This means that we cannot exclude A as possible subsumer for other (atomic) concepts if the nodes in the completion graph (or in the saturation graph) do not contain A, which is an important optimisation for classification procedures (cf. Section 4.2).

[4] Note that this only works as long as there are no other axioms of the form $A \sqsubseteq D'$ or $A \equiv D'$ with $D' \neq D$ in the knowledge base.

To compensate this, we can create a "candidate concept" A^+ for A, for example by partially absorbing D, which is then automatically added to a node in the completion graph if the node is possibly an instance of A. Hence, if A^+ is not added to a node label, then we know that A is not a possible subsumer of the concepts in the label of this node. Although the candidate concepts already allow a significant pruning of subsumption tests, there are still ontologies where we have to add these candidate concepts to many node labels, especially if only a limited absorption of D is possible. Hence, A can still be a possible subsumer for many concepts.

The saturation graph can, however, again be used to further improve the identification of (more or less obvious) non-subsumptions. Basically, if a candidate concept A^+ for $A \equiv D$ is in the label of a node v in the completion graph, then we test whether we can merge v with the saturated node $v_{\dot{\neg}D}$. Since D is often a conjunction, we can also try to merge v with the representative node for a disjunct of $\dot{\neg}D$. If the "models" can be "merged" as defined below, then v is obviously not an instance of A.

Definition 4 (Model Merging). *Let $S = (V, E, \mathcal{L})$ be a fully saturated saturation graph and $G = (V', E', \mathcal{L}', \dot{\neq})$ be a fully expanded and clash-free completion graph for a knowledge base $\mathcal{K}$. A node $v \in V$ is* mergeable *with a node $v' \in V'$ if*

- *v is not critical, not nominal dependent, and not clashed;*
- *$\{C, \dot{\neg}C\} \cap (\mathcal{L}(v) \cup \mathcal{L}'(v')) = \emptyset$ for some concept C;*
- *if $\{A_1, A_2\} \subseteq (\mathcal{L}(v) \cup \mathcal{L}'(v'))$ and $(A_1 \sqcap A_2) \sqsubseteq C \in \mathcal{K}$, then $C \in (\mathcal{L}(v) \cup \mathcal{L}'(v'))$;*
- *if $\forall r.C \in \mathcal{L}(v)$ $(\leqslant m\, r.C \in \mathcal{L}(v))$, then $C \in \mathcal{L}'(w')$ $(\dot{\neg}C \in \mathcal{L}'(w'))$ for every r-neighbour w' of v';*
- *if $\forall r.C \in \mathcal{L}'(v')$ $(\leqslant m\, r.C \in \mathcal{L}'(v'))$, then $C \in \mathcal{L}(w)$ $(\dot{\neg}C \in \mathcal{L}'(w'))$ for every r-successor w of v.*

The conditions that guarantee that the models are mergeable can be checked very efficiently. Note that it is possible to relax some of the conditions. For instance, it is not necessary to enforce that v is not nominal dependent as long as we can ensure that there is no interaction with the generated completion graph. This can, for example, be guaranteed if the completion graph does not use nominals.

5 Saturation Improvements

Obviously, the tableau algorithm can benefit more from the saturation, if few nodes are critical. In the following, we present different approaches for improving the saturation by reducing the number of critical nodes.

5.1 Extending Saturation to more Language Features

One way to improve the saturation is to extend the rules to cover more language features, e.g., as in the consequence-based reasoning procedure for Horn-$\mathcal{SHIF}$ [10]. Although we cannot directly modify existing r-successors to support universal restrictions of the form $\forall r.C$, we can easily create a new r-successor whose label additionally contains C. By further removing the previous r-successor, the node is no longer critical

due to C1. Analogously, to (partially) support at-most restrictions of the form $\leqslant 1\, r.\top$, several r-successors can be merged into a new node. To completely cover the DL $\mathcal{EL}^{++}$, it would be necessary to integrate a more sophisticated handling of nominals. Currently, we are, however, not aware of real-world ontologies where this would result in significant improvements. Note that it is possible to limit the number of additionally created nodes for these extensions and to consider parts that are not handled as critical, thus the overhead of the saturation can be managed.

5.2 Improving Saturation with Results from Completion Graphs

As already mentioned, even if there is only one critical representative node for a nominal, all nominal dependent nodes have to be considered critical. Analogously, nodes with incompletely handled concepts (e.g., disjunctions) are considered critical and also all nodes that indirectly refer to other critical nodes, even if all concepts in their labels can be handled completely. Extending the saturation rules only also has its limits since we are not aware of saturation-based procedures that cover very expressive DLs such as $\mathcal{SROIQ}$. Hence, we can still get many critical nodes in knowledge bases that use unsupported features.

An approach to overcome this issue is to "patch" the saturation graph with results from fully expanded and clash-free completion graphs, e.g., from consistency or satisfiability checks. Roughly speaking, we replace the labels of critical nodes in the saturation graph with corresponding labels from a completion graph, where we know that they are completely handled. Applying the saturation rules again, then hopefully results in a saturation graph with less critical nodes. However, since the completion graph contains deterministically and non-deterministically derived consequences, we also have to distinguish them for the saturation. An interesting way to achieve this is to simultaneously manage two saturation graphs: one where only the deterministically derived concepts are added and a second one, where also the non-deterministically derived concepts and consequences are considered. If the non-deterministic consequences have only a locally limited influence, i.e., the non-deterministically added concepts propagate new consequences only to a limited number of ancestor nodes, then, by comparing both saturation graphs, we can possibly identify ancestor nodes that are not further influenced by non-deterministic consequences and, thus, do not have to be considered critical.

6 Implementation and Evaluation

We extended Konclude[5] [17] with the presented saturation procedure and optimisations. Konclude is a tableau-based reasoner for $\mathcal{SROIQ}$ [6] with extensions for the handling of nominal schemas [15]. It integrates many state-of-the-art optimisations such as lazy unfolding, dependency directed backtracking, caching, etc. Moreover, Konclude uses partial absorption in order to significantly reduce the non-determinism in ontologies, which makes Konclude very suitable for the integration of saturation procedures.

The saturation algorithm integrated in Konclude almost covers the DL Horn-$\mathcal{SRIF}$ by using the extensions described in Section 5.1 for universal restrictions and functional

[5] Available at `http://www.konclude.com/`

Table 2. Statistics of ontology metrics for the evaluated ontology repositories (Ø stands for average and M for median)

Repository	# Ontologies	Axioms		Classes		Properties		Individuals	
		Ø	M	Ø	M	Ø	M	Ø	M
Gardiner	276	6,143	95	1,892	16	36	7	90	3
NCBO BioPortal	403	25,561	1,068	7,617	339	47	13	1,782	0
NCIt	185	178,818	167,667	69,720	68,862	116	123	0	0
OBO Foundry	422	44,424	1,990	8,033	839	28	6	24,868	66
Oxford	383	74,248	4,249	8,789	544	52	13	18,798	12
TONES	200	7,697	337	2,907	100	28	5	66	0
Google Crawl	413	6,282	194	1,122	38	69	15	830	1
OntoCrawler	544	1,876	119	125	18	56	12	638	0
OntoJCrawl	1,680	5,848	218	1,641	43	29	8	810	0
Swoogle Crawl	1,635	2,529	109	420	21	26	8	888	0
ALL	6,141	18,583	252	4,635	50	39	9	3,674	0

at-most restrictions (only merging with predecessors is not implemented). The number of nodes that are additionally processed for the handling of these saturation extensions is mainly limited by the number of concepts occurring in the knowledge base. However, the saturation in Konclude only supports a very limited handling of individuals since the individuals also have to be handled by the tableau algorithm (at least in the worst-case) and several representations of the individuals easily multiply the memory consumption. To compensate this, Konclude primarily handles individuals with the tableau algorithm and uses patches from completion graphs (as presented in Section 5.2) to improve those parts in the saturation graph that depend on nominals.

In the following, we present a detailed evaluation that shows the improvement of Konclude due to the integrated saturation procedure. The evaluation uses a large test corpus of ontologies which have been obtained by collecting all downloadable and parseable ontologies from the Gardiner ontology suite [4], the NCBO BioPortal,[6] the National Cancer Institute thesaurus (NCIt) archive,[7] the Open Biological Ontologies (OBO) Foundry [13], the Oxford ontology library,[8] the TONES repository,[9] and those subsets of the OWLCorpus [11] that were gathered by the crawlers Google, OntoCrawler, OntoJCrawl, and Swoogle.[10] All ontologies were parsed and converted to self-contained OWL/XML files with the OWL API. For the 1,380 ontologies with imports we created a version with resolved imports and another one without the imports (for testing the reasoning performance on the main ontology content without imports, which are frequently shared by many ontologies). Since Konclude does not yet support datatypes, we removed all data properties and we replaced all data property restrictions

[6] `http://bioportal.bioontology.org/`

[7] `http://ncit.nci.nih.gov/`

[8] `http://www.cs.ox.ac.uk/isg/ontologies/`; We ignored repositories that are redundantly contained in the Oxford ontology library (e.g., the Gardiner ontology suite).

[9] `http://owl.cs.manchester.ac.uk/repository/`

[10] In order to avoid too many redundant ontologies, we only used those subsets of the OWLCorpus which were gathered with the crawlers OntoCrawler, OntoJCrawl, Swoogle, and Google.

with *owl*:*Thing* in all ontologies. Table 2 shows an overview of our obtained test corpus with overall 6,141 ontologies including statistics of ontology metrics for the source repositories. Please note that 34.9 % of all ontologies are not even in the OWL 2 DL profile, which is, however, mainly due to undeclared entities.

The evaluation was carried out on a Dell PowerEdge R420 server running with two Intel Xeon E5-2440 hexa core processors at 2.4 GHz with Hyper-Threading and 48 GB RAM under a 64bit Ubuntu 12.04.2 LTS. Our evaluation focuses on classification, which is a central reasoning task that is supported by many reasoners and, thus, it is ideal for the comparison of results. In principle, we only measured the wall clock time for classification, i.e., the times spent for parsing and loading ontologies as well as for writing classification output to files are not included. Each test was executed with a time limit of 5 minutes, but without any limitation of memory allocation. Although Konclude supports parallelisation, we only used one worker thread, which allows for a comparison independent of the number of CPU cores and facilitates the presentation of the improvements through saturation.

Table 3 shows a comparison of the accumulated classification times for the evaluated repositories (in seconds) between the following versions of Konclude:

- NONE, where none of the saturation optimisations are activated,
- NONE+RT, where only the transfer of results from the saturation into the completion graph (as presented in Section 4.1) is activated;
- NONE+SE, where only the extraction of subsumers from the saturation (as presented in Section 4.2) is activated;
- NONE+MM, where only the model merging with the saturation graph (as presented in Section 4.3) is activated;
- ALL−SI, where the saturation improvements (as presented in Section 5) are deactivated (i.e., all the saturation optimisations presented in Setion 4 are activated),
- ALL, where all saturation optimisations and saturation improvements are activated.

In addition, the column on the right side shows the performance gains (in percent) from NONE to the version ALL. Please note that the saturation improvements are optimisations to further improve the saturation procedure and, therefore, a separate evaluation of these techniques does not make sense.

It can be observed that the most significant improvements are achieved with the model merging optimisation (cf. NONE+MM), which is due to the large amount of NCI-Thesaurus ontologies in the NCIt archive, where this optimisation significantly reduces the classification effort. In contrast, if only the transfer of the saturation results (NONE+RT) or the extraction of subsumers from the saturation (NONE+SE) is activated, then only minor improvements with respect to the version NONE are possible. However, the combined activation of these optimisations (cf. ALL−SI) again leads to a significant performance gain for the repositories, which indicates that there is a synergy effect from the combination of these optimisation. Since all of these optimisations are based on the saturation procedure, which also requires a significant amount of processing time for large ontologies (approximately $1,953$ s for all repositories), this synergy effect is not very surprising. By further activating the saturation improvements (cf. ALL), we obtain another performance gain. Considering all repositories, the

Table 3. Accumulated classification times (in seconds) with separately activated saturation optimisations for the evaluated ontology repositories

Repository	NONE	NONE+RT	NONE+SE	NONE+MM	ALL−SI	ALL	↓ [%]
Gardiner	531	611	469	535	558	559	−5.2
NCBO BioPortal	2,071	1,947	971	2,156	988	793	61.7
NCIt	28,639	28,538	28,276	3,223	2,496	2,457	91.4
OBO Foundry	879	821	979	1,078	741	649	26.2
Oxford	6,623	5,006	6,012	6,510	3,429	2,743	58.6
TONES	1,756	1,456	1,413	494	321	337	80.8
Google Crawl	465	428	448	467	363	138	70.3
OntoCrawler	26	25	24	25	23	22	14.7
OntoJCrawl	1,417	923	715	1,427	517	548	61.4
Swoogle Crawl	2,501	2,502	2,493	1,402	1,248	1,343	46.3
ALL	44,910	42,256	41,800	17,317	10,684	9,589	78.6

combined activation of all saturation optimisations and improvements reduces the accumulated reasoning time by 78.6 %. It is also worth pointing out that the version NONE timed out for 128 ontologies, whereas the version ALL only reached the time limit for 10 ontologies. Thus, an evaluation with an increased time limit would show even better performance gains. For example, the Oxford ontology library contains the SCT-SEP ontology,[11] which can be classified by the version ALL in 181.2 s, whereas the version NONE requires 1709.4 s. SCT-SEP is a SNOMED extension that intensively uses disjunctions and disjointness and, thus, is clearly outside the OWL EL fragment. Nevertheless, large parts of the ontology have an EL structure and, therefore, our optimisations are able to improve the reasoning performance by almost one order of magnitude.

Table 3 also reveals that some saturation optimisations are not really relevant for some repositories. For instance, the activated result transfer yields worse reasoning times for the ontologies in the Gardiner ontology suite. Moreover, the model merging optimisation causes significant performance losses for some repositories (e.g., for OBO Foundry), which indicates that further optimisation is possible; e.g., one could learn statistics about the success of model merging with certain nodes in the saturation graph and automatically skip a merging test if there is a high likelihood that it will fail.

7 Conclusions

In this paper, we have presented a technique for tightly coupling saturation- and tableau-based procedures. Unlike standard consequence-based procedures, the approach is applicable on arbitrary OWL 2 DL ontologies. Furthermore, it has a very good pay-as-you-go behaviour, i.e., if only few axioms use features that are problematic for saturation-based procedures (e.g., disjunction), then the tableau procedure can still benefit significantly from the saturation. This seems to be confirmed by our evaluation over several thousand ontologies, where the integration of the presented saturation optimisations into the reasoning system Konclude significantly improves the classification performance.

[11] Originally from `https://code.google.com/p/condor-reasoner/`

References

1. Armas Romero, A., Cuenca Grau, B., Horrocks, I.: MORe: Modular combination of OWL reasoners for ontology classification. In: Proc. 11th Int. Semantic Web Conf. (ISWC'12). LNCS, vol. 7649, pp. 1–16. Springer (2012)
2. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proc. 19th Int. Joint Conf. on Artificial Intelligence (IJCAI'05). pp. 364–369. Professional Book Center (2005)
3. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, 2nd edn. (2007)
4. Gardiner, T., Horrocks, I., Tsarkov, D.: Automated benchmarking of description logic reasoners. In: Proc. 19th Int. Workshop on Description Logics (DL'06). vol. 198. CEUR (2006)
5. Glimm, B., Horrocks, I., Motik, B., Shearer, R., Stoilos, G.: A novel approach to ontology classification. J. of Web Semantics 14, 84–101 (2012)
6. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: Proc. 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'06). pp. 57–67. AAAI Press (2006)
7. Horrocks, I., Tobies, S.: Reasoning with axioms: Theory and practice. In: Proc. 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'00). pp. 285–296. Morgan Kaufmann (2000)
8. Hudek, A.K., Weddell, G.E.: Binary absorption in tableaux-based reasoning for description logics. In: Proc. 19th Int. Workshop on Description Logics (DL'06). vol. 189. CEUR (2006)
9. Kazakov, Y.: $\mathcal{RIQ}$ and $\mathcal{SROIQ}$ are harder than $\mathcal{SHOIQ}$. In: Proc. 11th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'08). pp. 274–284. AAAI Press (2008)
10. Kazakov, Y.: Consequence-driven reasoning for Horn-$\mathcal{SHIQ}$ ontologies. In: Proc. 21st Int. Conf. on Artificial Intelligence (IJCAI'09). pp. 2040–2045. IJCAI (2009)
11. Matentzoglu, N., Bail, S., Parsia, B.: A corpus of OWL DL ontologies. In: Proc. 26th Int. Workshop on Description Logics (DL'13). vol. 1014. CEUR (2013)
12. Simančík, F., Kazakov, Y., Horrocks, I.: Consequence-based reasoning beyond Horn ontologies. In: Proc. 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI'11). pp. 1093–1098. IJCAI/AAAI (2011)
13. Smith, B., Ashburner, M., Rosse, C., Bard, J., Bug, W., Ceusters, W., Goldberg, L.J., Eilbeck, K., Ireland, A., Mungall, C.J., The OBI Consortium, Leontis, N., Rocca-Serra, P., Ruttenberg, A., Sansone, S.A., Scheuermann, R.H., Shah, N., Whetzeland, P.L., Lewis, S.: The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. Nature Biotechnology 25, 1251–1255 (2007)
14. Song, W., Spencer, B., Du, W.: WSReasoner: A prototype hybrid reasoner for $\mathcal{ALCHOI}$ ontology classification using a weakening and strengthening approach. In: Proc. 1st Int. Workshop on OWL Reasoner Evaluation (ORE'12). vol. 858. CEUR (2012)
15. Steigmiller, A., Glimm, B., Liebig, T.: Nominal schema absorption. In: Proc. 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI'13). pp. 1104–1110. AAAI Press (2013)
16. Steigmiller, A., Glimm, B., Liebig, T.: Coupling tableau algorithms for the DL $\mathcal{SROIQ}$ with completion-based saturation procedures. Tech. Rep. UIB-2014-02, University of Ulm, Ulm, Germany (2014), available online at `http://www.uni-ulm.de/fileadmin/website_uni_ulm/iui/Ulmer_Informatik_Berichte/2014/UIB-2014-02.pdf`
17. Steigmiller, A., Liebig, T., Glimm, B.: Konclude: system description. J. of Web Semantics (2014), accepted
18. Tsarkov, D., Horrocks, I., Patel-Schneider, P.F.: Optimizing terminological reasoning for expressive description logics. J. of Automated Reasoning 39, 277–316 (2007)
19. W3C OWL Working Group: OWL 2 Web Ontology Language: Document Overview. W3C Recommendation (27 October 2009)