Journal of Artificial Intelligence Research 54 (2015) 535-592 Submitted 07/15; published 12/15

Pay-As-You-Go Description Logic Reasoning by Coupling Tableau and Saturation Procedures

Andreas Steigmiller ANDREAS.STEIGMILLER@UNI-ULM.DE
Birte Glimm BIRTE.GLIMM@UNI-ULM.DE
Institute of Artificial Intelligence, University of Ulm, Germany

Abstract

Nowadays, saturation-based reasoners for the OWL EL profile of the Web Ontology Language are able to handle large ontologies such as SNOMED very efficiently. However, it is currently unclear how saturation-based reasoning procedures can be extended to very expressive Description Logics such as $\mathcal{SROIQ}$—the logical underpinning of the current and second iteration of the Web Ontology Language. Tableau-based procedures, on the other hand, are not limited to specific Description Logic languages or OWL profiles, but even highly optimised tableau-based 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 for a wide range of ontologies.

1. Introduction

The current version of the Web Ontology Language (OWL 2) (W3C OWL Working Group, 2009) is based on the very expressive Description Logic (DL) $\mathcal{SROIQ}$ (Horrocks, Kutz, & Sattler, 2006). Sound and complete tableau algorithms, which are easily extensible and adaptable, are typically used to handle (standard) reasoning tasks. 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 (Kazakov, 2008), it is, however, not surprising that larger ontologies easily become impractical for existing systems.

In contrast, the OWL 2 profiles define language fragments of $\mathcal{SROIQ}$ for which the standard reasoning tasks are tractable and specialised reasoning procedures are available. For example, the OWL 2 EL profile is based on the DL $\mathcal{EL}^{++}$, which can efficiently be handled by completion- or consequence-based reasoning procedures (Baader, Brandt, & Lutz, 2005; Kazakov, 2009). These saturation algorithms have also been extended to handle more expressive DLs such as Horn-$\mathcal{SHIQ}$ (Kazakov, 2009) for which they are often able to outperform the more general tableau algorithms. Even saturation procedures for DLs with non-deterministic language features, such as $\mathcal{ALCI}$ (Simančík, Kazakov, & Horrocks, 2011), $\mathcal{ALCH}$ (Simančík, Motik, & Horrocks, 2014), $\mathcal{SHIQ}$ (Bate, Motik, Cuenca Grau, Simančík, & Horrocks, 2015), have been developed and some of their implementations into reasoning systems show a remarkable performance. In particular, they allow for a one-pass handling of several reasoning tasks such as classification (i.e., the task of arranging the classes of an ontology in a hierarchy), whereas the idea of tableau procedures is based on pairwise testing individual class subsumptions. Although optimised classification algorithms have

been developed for tableau-based reasoning systems (Baader, Hollunder, Nebel, Profitlich, & Franconi, 1994; Glimm, Horrocks, Motik, Shearer, & Stoilos, 2012), they still use a multitude of separate consistency tests in order to decide subsumption relations. Since a complete handling of DLs providing disjunctions, cardinality restrictions, and inverse roles causes several difficulties, saturation procedures have not yet been extended to very expressive DLs such as $\mathcal{SROIQ}$. Hence, only the less efficient tableau-based reasoning systems can currently be used to handle ontologies with these more expressive language features.

Unfortunately, a combination of tableau algorithms and saturation procedures is not straightforward since both techniques work quite differently. Hence, ontology engineers have to decide whether they only use the restricted features of certain language fragments such that their ontologies can be handled by specialised reasoners with saturation-based procedures or they have to face possible performance losses by using more general reasoning systems based on tableau algorithms. This is especially unfavourable if such language features are only required for a few axioms in the ontology. In this case, completeness is no longer ensured for the specialised procedures, while the fully-fledged, tableau-based reasoners are possibly not efficient enough. Ideally, reasoning systems with a better pay-as-you-go behaviour could be used, where the part of the ontology that is not affected by the axioms outside the tractable fragment can still be handled efficiently. This led to the recent development of approaches that combine saturation procedures and fully-fledged reasoners in a black box manner (Armas Romero, Cuenca Grau, & Horrocks, 2012; Song, Spencer, & Du, 2012; Zhou, Nenov, Cuenca Grau, & Horrocks, 2014; Zhou, Cuenca Grau, Nenov, Kaminski, & Horrocks, 2015). 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, while still guaranteeing completeness.

In this paper, we present a much tighter coupling between saturation and tableau 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 very 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 discuss related work (Section 6) and present the results of a detailed evaluation including comparisons with other approaches and state-of-the-art reasoners (Section 7) before we conclude (Section 8).

This paper is based on a previous conference publication (Steigmiller, Glimm, & Liebig, 2014a), but contains significantly extended explanations, additional examples, and proofs for the correctness of the integrated saturation procedure with its incompleteness detection. Due to the space limitations of the conference publication, the information passing between the tableau algorithm and the saturation procedure could only be sketched, whereas it is described in detail in this paper. Moreover, the coupling with the saturation procedure has been extended in this paper to consider and handle all language features of the DL

$\mathcal{SROIQ}$. Furthermore, we show a new and more detailed evaluation that is based on an updated implementation and we compare the results with other approaches where fully-fledged and specialised reasoners are combined.

2. Preliminaries

In order to describe our techniques in more detail, we first give, based on the original presentation of Horrocks et al. (2006), a brief introduction into the DL $\mathcal{SROIQ}$ in this section. For a detailed introduction to DLs, we refer to the Description Logic Handbook (Baader, Calvanese, McGuinness, Nardi, & Patel-Schneider, 2007). Subsequently, we describe a tableau algorithm as it is typically used by reasoning systems and also refer to the work of Horrocks et al. (2006) for further details.

2.1 The Description Logic $\mathcal{SROIQ}$

We first define the syntax of roles, concepts (also called classes), and individuals, and then we go on to axioms and ontologies/knowledge bases. Additionally, we sketch typically used restrictions for the combination of the different axioms, which are necessary to ensure decidability for many inference problems of $\mathcal{SROIQ}$. Note that we do not describe all details of these restrictions since they are well-known in the DL literature (Horrocks et al., 2006) but not particularly relevant for the proposed optimisation techniques. Subsequently, we define the semantics of these components.

Definition 1 (Syntax of Individuals, Concepts, and Roles)**.** *Let N_C, N_R, and N_I be countable, infinite, and pairwise disjoint sets of* concept names*,* role names*, and* individual names*, respectively. We call $\Sigma = (N_C, N_R, N_I)$ a* signature*. The set $\mathsf{Rols}(\Sigma)$ of $\mathcal{SROIQ}$-*roles *over Σ (or roles for short) is $N_R \cup \{r^- \mid r \in N_R\}$, where a role of the form r^- is called the* inverse role *of r. Since the inverse relation on roles is symmetric, we can define a function inv, which returns the inverse of a role and, therefore, we do not have to consider roles of the from r^{--}. For $r \in N_R$, let be $\mathsf{inv}(r) = r^-$ and $\mathsf{inv}(r^-) = r$.*

*The set of $\mathcal{SROIQ}$-*concepts *(or concepts for short) over Σ is the smallest set built inductively over symbols from Σ using the following grammar, where $a \in N_I, n \in \mathbb{N}_0, A \in N_C$, and $r \in \mathsf{Rols}(\Sigma)$:*

$$C ::= \quad \top \mid \bot \mid A \mid \{a\} \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \forall r.C \mid \exists r.C \mid \exists r.\mathsf{Self} \mid \geqslant n\, r.C \mid \leqslant n\, r.C.$$

The roles, concepts, and individuals are used to build ontology axioms as follows:

Definition 2 (Syntax of Axioms and Ontologies)**.** *For C, D concepts, a* general concept inclusion *(GCI) axiom is an expression $C \sqsubseteq D$. A finite set of GCIs is called a* TBox*.*

A role inclusion *(RI) axiom is an expression of the form $u \sqsubseteq r$, where r is a role and u is a composition of roles, i.e., $u = s_1 \circ \ldots \circ s_n$ with the roles $s_1, \ldots, s_n$ and $n \geq 1$. For r, s roles, a* role assertion *(RA) axiom is of the form $\mathsf{Disj}(r, s)$. An RBox is a finite set of RIs and RAs. Given an RBox $\mathcal{R}$, we use $\sqsubseteq^*$ as the transitive-reflexive closure over all $r \sqsubseteq s$ and $\mathsf{inv}(r) \sqsubseteq \mathsf{inv}(s)$ axioms in $\mathcal{R}$. We call a role r a* sub-role *of s and s a* super-role *of r if $r \sqsubseteq^* s$.*

An (ABox) assertion *is an expression of the form $C(a)$ or $r(a, b)$, where C is a concept, r is a role, and $a, b \in N_I$ are individual names. An* ABox *is a finite set of assertions. A*

knowledge base *or* ontology $\mathcal{K}$ *is a tuple* $(\mathcal{T}, \mathcal{R}, \mathcal{A})$ *with* $\mathcal{T}$ *a TBox,* $\mathcal{R}$ *an RBox, and* $\mathcal{A}$ *an ABox.*

Note, it is also possible to allow other types of RAs for the RBox, e.g., axioms that specify roles as asymmetric, irreflexive, transitive, reflexive, or symmetric. However, such axioms can be expressed indirectly in other ways and, therefore, we omit their presentation here. For example, an axiom of the form $\mathsf{Refl}(r)$, for which the role r has to be interpreted as reflexive, can be encoded with the axioms $r' \sqsubseteq r$ and $\top \sqsubseteq \exists r'.\mathsf{Self}$.[1] Analogously, we only allow the most frequently used ABox assertions since, in the presence of nominals, all ABox assertion can also be expressed with GCIs (which we also utilise below to eliminate all ABox assertions to simplify the presentation of algorithms). Furthermore, $\mathcal{SROIQ}$ usually allows the usage of the universal role u, but u can also be simulated by a fresh transitive, reflexive, and symmetric super role, i.e., a role that is implied by all other roles. In the following, we use $\mathcal{K}$ also as an abbreviation for the collection of all axioms in the knowledge base. For example, we write $C \sqsubseteq D \in \mathcal{K}$ instead of $C \sqsubseteq D \in \mathcal{T}$ and $\mathcal{T} \in \mathcal{K}$.

In order to ensure decidability (Horrocks, Sattler, & Tobies, 1999; Horrocks & Sattler, 2004), only simple roles are allowed in concepts of the form $\geqslant n\, r.C$, $\leqslant n\, r.C$, and $\exists r.\mathsf{Self}$ and in axioms of the form $\mathsf{Disj}(r, s)$, where, roughly speaking, a role is simple if it is not implied by any RI that uses role composition. Furthermore, the RBox has to be regular, i.e., RI axioms are only allowed in a limited form (Horrocks & Sattler, 2004), which restricts cyclic dependencies between RIs.

Next, we define the semantics of concepts and then we go on to the semantics of axioms and ontologies/knowledge bases.

Definition 3 (Semantics of Individuals, Concepts, and Roles)**.** *An* interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *consists of a non-empty set* $\Delta^{\mathcal{I}}$*, the* domain *of* $\mathcal{I}$*, and a function* $\cdot^{\mathcal{I}}$*, which maps every concept name* $A \in N_C$ *to a subset* $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$*, every role name* $r \in N_R$ *to a binary relation* $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$*, and every individual name* $a \in N_I$ *to an element* $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$*. For each role name* $r \in N_R$*, the interpretation of its inverse role* $(r^-)^{\mathcal{I}}$ *consists of all pairs* $\langle \delta, \delta' \rangle \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ *for which* $\langle \delta', \delta \rangle \in r^{\mathcal{I}}$*.*

For any interpretation $\mathcal{I}$*, the semantics of* $\mathcal{SROIQ}$*-concepts over a signature* Σ *is defined by the function* $\cdot^{\mathcal{I}}$ *as follows:*

$$
\begin{array}{rlrlrl}
\top^{\mathcal{I}} = & \Delta^{\mathcal{I}} & \bot^{\mathcal{I}} = & \emptyset & (\{a\})^{\mathcal{I}} = & \{a^{\mathcal{I}}\} \\
(\neg C)^{\mathcal{I}} = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} & (C \sqcap D)^{\mathcal{I}} = & C^{\mathcal{I}} \cap D^{\mathcal{I}} & (C \sqcup D)^{\mathcal{I}} = & C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\exists r.\mathsf{Self})^{\mathcal{I}} = & \{\delta \in \Delta^{\mathcal{I}} \mid \langle \delta, \delta \rangle \in r^{\mathcal{I}}\} & & & & \\
(\forall r.C)^{\mathcal{I}} = & \{\delta \in \Delta^{\mathcal{I}} \mid \textit{if } \langle \delta, \delta' \rangle \in r^{\mathcal{I}}, \textit{ then } \delta' \in C^{\mathcal{I}}\} & & & & \\
(\exists r.C)^{\mathcal{I}} = & \{\delta \in \Delta^{\mathcal{I}} \mid \textit{there is a } \langle \delta, \delta' \rangle \in r^{\mathcal{I}} \textit{ with } \delta' \in C^{\mathcal{I}}\} & & & & \\
(\leqslant n\, r.C)^{\mathcal{I}} = & \{\delta \in \Delta^{\mathcal{I}} \mid \sharp\{\delta' \in \Delta^{\mathcal{I}} \mid \langle \delta, \delta' \rangle \in r^{\mathcal{I}} \textit{and } \delta' \in C^{\mathcal{I}}\} \leq n\} & & & & \\
(\geqslant n\, r.C)^{\mathcal{I}} = & \{\delta \in \Delta^{\mathcal{I}} \mid \sharp\{\delta' \in \Delta^{\mathcal{I}} \mid \langle \delta, \delta' \rangle \in r^{\mathcal{I}} \textit{and } \delta' \in C^{\mathcal{I}}\} \geq n\}, & & & & \\
\end{array}
$$

where $\sharp M$ *denotes the cardinality of the set* M*.*

Finally, we can define the semantics of ontologies/knowledge bases.

1. Note that we have to use the fresh sub-role r' of r in the axiom $\top \sqsubseteq \exists r'.\mathsf{Self}$ since r might be complex, but $\exists r.\mathsf{Self}$ expressions are only allowed with simple roles.

Definition 4 (Semantics of Axioms and Ontologies)**.** *Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation, then $\mathcal{I}$* satisfies *a TBox/RBox axiom or ABox assertion α, written $\mathcal{I} \models \alpha$ if*

1. *α is a GCI $C \sqsubseteq D$ and $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, or*
2. *α is an RI $s_1 \circ \ldots \circ s_n \sqsubseteq r$ and $s_1^{\mathcal{I}} \circ \ldots \circ s_n^{\mathcal{I}} \subseteq r^{\mathcal{I}}$, where $\circ$ denotes the composition of binary relations for $s_1^{\mathcal{I}} \circ \ldots \circ s_n^{\mathcal{I}}$, or*
3. *α is an RA of the form $\mathsf{Disj}(r, s)$ and $r^{\mathcal{I}} \cap s^{\mathcal{I}} = \emptyset$, or*
4. *α is an ABox assertion $C(a)$ and $a^{\mathcal{I}} \in C^{\mathcal{I}}$, or*
5. *α is an ABox assertion $r(a, b)$ and $\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in r^{\mathcal{I}}$.*

$\mathcal{I}$ satisfies *a TBox $\mathcal{T}$ (RBox $\mathcal{R}$, ABox $\mathcal{A}$) if it satisfies each GCI in $\mathcal{T}$ (each RI/RA axiom in $\mathcal{R}$, each assertion in $\mathcal{A}$). We say that $\mathcal{I}$* satisfies *$\mathcal{K} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ if $\mathcal{I}$ satisfies $\mathcal{T}$, $\mathcal{R}$, and $\mathcal{A}$. In this case, we say that $\mathcal{I}$ is a* model *of $\mathcal{K}$ and we write $\mathcal{I} \models \mathcal{K}$. We say that $\mathcal{K}$* is consistent *if $\mathcal{K}$ has a model.*

2.2 Normalisation and Preprocessing

In the remainder we assume that each knowledge base is normalised into the following form:

1. The TBox contains only axioms of the form $A_1 \sqcap A_2 \sqsubseteq C$ and $H \sqsubseteq C$ with $H = A$, $H = \{a\}$, or $H = \top$, where C is in negation normal form and A, A_1, A_2 denote concept names.
2. The RBox contains only RAs and simple role inclusion axioms of the form $r_1 \sqsubseteq r_2$.
3. The ABox is empty.

This assumption is without loss of generality. For Item 1: Any concept can be transformed into an equivalent one in negation normal form (NNF) by pushing negation inwards, making use of de Morgan's laws and the duality between existential and universal restrictions, and between at-most and at-least cardinality restrictions (Horrocks, Sattler, & Tobies, 2000). We use $\mathsf{nnf}(C)$ to denote the equivalent concept to C in NNF. Furthermore, a GCI $C \sqsubseteq D \in \mathcal{T}$ that does not correspond to the normal form can equivalently be written as $\top \sqsubseteq \mathsf{nnf}(\neg C \sqcup D)$. This rewriting of GCIs creates (possibly many) disjunctions, which potentially causes a lot of non-determinism in the reasoning procedure and, therefore, easily decreases the reasoning performance. To counteract this, a preprocessing step called *absorption* is often used (Horrocks & Tobies, 2000; Hudek & Weddell, 2006; Steigmiller, Glimm, & Liebig, 2013, 2014b; Tsarkov & Horrocks, 2004), which tries to rewrite the axioms into possibly several simpler concept inclusion axioms. For example, instead of treating $A \sqcap \exists r.B \sqsubseteq C$ as $\top \sqsubseteq \neg A \sqcup \forall r.(\neg B) \sqcup C$, a sophisticated absorption algorithm can avoid the non-determinism by rewriting the axiom into $B \sqsubseteq \forall r^{-}.F$ and $A \sqcap F \sqsubseteq C$, where F is a fresh atomic concept that is used to preserve the semantics of the original axiom. For Item 2: RIs that use compositions can be eliminated using an encoding based on automata (Horrocks & Sattler, 2004) or regular expressions (Simančík, 2012). Note that such explicit encodings of propagations over complex roles might blow up the knowledge base exponentially, but

it cannot be avoided in the worst-case, i.e., we could only try to delay it until the actual reasoning process (Kazakov, 2008) and this is indeed utilised by many reasoners (although such a blow up seems hardly be caused by real-world ontologies). For Item 3, $C(a)$ ($r(a,b)$) can equivalently be expressed as $\{a\} \sqsubseteq C$ ($\{a\} \sqsubseteq \exists r.\{b\}$).

2.3 Tableau Algorithm for $\mathcal{SROIQ}$

Model construction calculi, such as tableaux, decide 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". In the following, we describe, based on the original presentation of the $\mathcal{SROIQ}$ tableau algorithm (Horrocks et al., 2006), the model construction process and the used data structures, beginning with completion graphs.

Definition 5 (Completion Graph). *For a concept C, we use $\mathsf{sub}(C)$ to denote the set of all sub-concepts of C (including C). Let $\mathcal{K}$ be a normalised $\mathcal{SROIQ}$ knowledge base and let $\mathsf{Cons}(\mathcal{K})$ be the set of concepts occurring in the TBox $\mathcal{T}$ of $\mathcal{K}$, i.e., $\mathsf{Cons}(\mathcal{K}) = \{C, D \mid C \sqsubseteq D \in \mathcal{K}\}$. We define the* closure $\mathsf{clos}(\mathcal{K})$ of $\mathcal{K}$ *as:*

$$\mathsf{clos}(\mathcal{K}) = \{C \in \mathsf{sub}(D) \mid D \in \mathsf{Cons}(\mathcal{K})\} \cup \{\mathsf{nnf}(\neg C) \mid C \in \mathsf{sub}(D), D \in \mathsf{Cons}(\mathcal{K})\}.$$

A completion graph *for $\mathcal{K}$ is a directed graph $G = (V, E, \mathcal{L}, \dot{\neq})$. Each node $v \in V$ is labelled with a set $\mathcal{L}(v) \subseteq \mathsf{fclos}(\mathcal{K})$, where*

$$\mathsf{fclos}(\mathcal{K}) = \mathsf{clos}(\mathcal{K}) \cup \{\leqslant m\, r.C \mid \leqslant n\, r.C \in \mathsf{clos}(\mathcal{K}) \text{ and } m \leq n\}.$$

Each edge $\langle v, v' \rangle \in E$ is labelled with the set $\mathcal{L}(\langle v, v' \rangle) \subseteq \mathsf{Rols}(\mathcal{K})$, where $\mathsf{Rols}(\mathcal{K})$ are the roles occurring in $\mathcal{K}$. The symmetric binary relation $\dot{\neq}$ is used to keep track of inequalities between nodes in V.

In the following, we often use $r \in \mathcal{L}(\langle v_1, v_2 \rangle)$ as an abbreviation for $\langle v_1, v_2 \rangle \in E$ and $r \in \mathcal{L}(\langle v_1, v_2 \rangle)$.

Definition 6 (Successor, Predecessor, Neighbour). *If $\langle v_1, v_2 \rangle \in E$, then v_2 is called a* successor *of v_1 and v_1 is called a* predecessor *of v_2.* Ancestor *is the transitive closure of predecessor, and* descendant *is the transitive closure of successor. A node v_2 is called an s-*successor *of a node v_1 if $r \in \mathcal{L}(\langle v_1, v_2 \rangle)$ and r is a sub-role of s; v_2 is called an s-*predecessor *of v_1 if v_1 is an s-successor of v_2. A node v_2 is called a neighbour (s-neighbour) of a node v_1 if v_2 is a successor (s-successor) of v_1 or if v_1 is a successor ($\mathsf{inv}(s)$-successor) of v_2.*

For a role r and a node $v \in V$, we define the set of v's r-neighbours with the concept C in their label, written $\mathsf{mneighbs}(v, r, C)$, as $\{v' \in V \mid v'$ is an r-neighbour of v and $C \in \mathcal{L}(v')\}$.

Note, many inference problems for the DL $\mathcal{SROIQ}$ can easily be reduced to consistency checking and, therefore, they can indirectly be handled although often only consistency checking reasoning task is specified for the tableau algorithm. For example, in order to test the satisfiability of a concept C, we introduce a fresh individual a for which we assert the concept C by an axiom of the form $\{a\} \sqsubseteq C$. The nodes that represent these (fresh) individuals are typically called *root nodes*.

In order to test the consistency of a knowledge base, the completion graph is initialised by creating one node for each individual/nominal in the input knowledge base. In particular,

if $v_1, \ldots, v_\ell$ are the nodes for the individuals $a_1, \ldots, a_\ell$ of $\mathcal{K}$, then we create an initial completion graph $G = (\{v_1, \ldots, v_\ell\}, \emptyset, \mathcal{L}, \emptyset)$ and add for each individual a_i the nominal $\{a_i\}$ and the concept $\top$ to the label of v_i, i.e., $\mathcal{L}(v_i) = \{\{a_i\}, \top\}$ for all $1 \leq i \leq \ell$.

The tableau algorithm works by decomposing/unfolding concepts in the completion graph with a set of expansion rules (see Table 1). Each rule application can add new concepts to node labels and/or new nodes and edges to the completion graph, thereby explicating the structure of a model for the input knowledge base. The rules are repeatedly 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 $\neg C$ 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 and clash-free completion graph.

Definition 7 (Clash)**.** *A completion graph $G = (V, E, \mathcal{L}, \dot{\neq})$ for a knowledge base $\mathcal{K}$ contains a clash if there are the nodes v and w such that*

1. $\bot \in \mathcal{L}(v)$, *or*
2. $\{C, \mathsf{nnf}(\neg C)\} \subseteq \mathcal{L}(v)$ *for some concept C, or*
3. *v is an r-neighbour of v and* $\neg\exists r.\mathsf{Self} \in \mathcal{L}(v)$, *or*
4. $\mathsf{Disj}(r, s) \in \mathcal{K}$ *and w is an r- and an s-neighbour of v, or*
5. *there is some concept* $\leqslant n\, r.C \in \mathcal{L}(v)$ *and* $\{w_1, \ldots, w_{n+1}\} \subseteq \mathsf{mneighbs}(v, r, C)$ *with* $w_i \dot{\neq} w_j$ *for all* $1 \leq i < j \leq n+1$, *or*
6. *there is some* $\{a\} \in \mathcal{L}(v) \cap \mathcal{L}(w)$ *and* $v \dot{\neq} w$.

Unrestricted application of the $\exists$-rule and $\geqslant$-rule can lead to the introduction of infinitely many new tableau nodes and, thus, prevent the calculus from terminating. To counteract that, a cycle detection technique called *(pairwise) blocking* (Horrocks & Sattler, 1999) is used that restricts the application of these rules. To apply blocking, we distinguish *blockable nodes* from *nominal nodes*, which have either an original nominal from the knowledge base or a new nominal introduced by the calculus in their label.

Definition 8 (Pairwise Blocking)**.** *A node is* blocked *if either it is directly or indirectly blocked. A node v is* indirectly blocked *if an ancestor of v is blocked; and v with predecessor v' is* directly blocked *if there exists an ancestor node w of v with predecessor w' such that*

1. *v, v', w, w' are all blockable,*
2. *w, w' are not blocked,*
3. $\mathcal{L}(v) = \mathcal{L}(w)$ *and* $\mathcal{L}(v') = \mathcal{L}(w')$,
4. $\mathcal{L}(\langle v', v\rangle) = \mathcal{L}(\langle w', w\rangle)$.

In this case, we say that w directly blocks v and w is the blocker of v.

$\sqsubseteq_1$-rule	if	$H \in \mathcal{L}(v)$, $H \sqsubseteq C \in \mathcal{K}$ with $H = A$, or $H = \{a\}$, or $H = \top$, $C \notin \mathcal{L}(v)$, and v is not indirectly blocked
	then	$\mathcal{L}(v) = \mathcal{L}(v) \cup \{C\}$
$\sqsubseteq_2$-rule	if	$\{A_1, A_2\} \subseteq \mathcal{L}(v)$, $A_1 \sqcap A_2 \sqsubseteq C \in \mathcal{K}$, $C \notin \mathcal{L}(v)$, and v is not indirectly blocked
	then	$\mathcal{L}(v) = \mathcal{L}(v) \cup \{C\}$
$\sqcap$-rule	if	$C_1 \sqcap C_2 \in \mathcal{L}(v)$, v is not indirectly blocked, and $\{C_1, C_2\} \not\subseteq \mathcal{L}(v)$
	then	$\mathcal{L}(v) = \mathcal{L}(v) \cup \{C_1, C_2\}$
$\sqcup$-rule	if	$C_1 \sqcup C_2 \in \mathcal{L}(v)$, v is not indirectly blocked, and $\{C_1, C_2\} \cap \mathcal{L}(v) = \emptyset$
	then	$\mathcal{L}(v') = \mathcal{L}(v') \cup \{H\}$ for some $H \in \{C_1, C_2\}$
$\exists$-rule	if	$\exists r.C \in \mathcal{L}(v)$, v is not blocked, and v has no r-neighbour v' with $C \in \mathcal{L}(v')$
	then	create a new node v' and an edge $\langle v, v' \rangle$ with $\mathcal{L}(v') = \{\top, C\}$ and $\mathcal{L}(\langle v, v' \rangle) = \{r\}$
Self-rule	if	$\exists r.\mathsf{Self} \in \mathcal{L}(v)$, v is not blocked, and v is no r-neighbour of v
	then	create a new edge $\langle v, v \rangle$ with $\mathcal{L}(\langle v, v \rangle) = \{r\}$
$\forall$-rule	if	$\forall r.C \in \mathcal{L}(v)$, v is not indirectly blocked, and there is an r-neighbour v' of v with $C \notin \mathcal{L}(v')$
	then	$\mathcal{L}(v') = \mathcal{L}(v') \cup \{C\}$
ch-rule	if	$\leqslant n\, r.C \in \mathcal{L}(v)$, v is not indirectly blocked, and there is an r-neighbour v' of v with $\{C, \mathsf{nnf}(\neg C)\} \cap \mathcal{L}(v') = \emptyset$
	then	$\mathcal{L}(v') = \mathcal{L}(v') \cup \{H\}$ for some $H \in \{C, \mathsf{nnf}(\neg C)\}$
$\geqslant$-rule	if	1. $\geqslant n\, r.C \in \mathcal{L}(v)$, v is not blocked, and 2. there are not n r-neighbours $v_1, \ldots, v_n$ of v with $C \in \mathcal{L}(v_i)$ and $v_i \dot{\neq} v_j$ for $1 \leq i < j \leq n$, where $v_1, \ldots, v_n$ are not blocked if v is a nominal node
	then	create n new nodes $v_1, \ldots, v_n$ with $\mathcal{L}(\langle v, v_i \rangle) = \{r\}$, $\mathcal{L}(v_i) = \{\top, C\}$ and $v_i \dot{\neq} v_j$ for $1 \leq i < j \leq n$.
$\leqslant$-rule	if	1. $\leqslant n\, r.C \in \mathcal{L}(v)$, v is not indirectly blocked, 2. $\sharp\mathsf{mneighbs}(v, r, C) > n$ and there are two r-neighbours v_1, v_2 of v with $C \in (\mathcal{L}(v_1) \cap \mathcal{L}(v_2))$ and not $v_1 \dot{\neq} v_2$
	then	a. if v_1 is a nominal node, then $\mathsf{merge}(v_2, v_1)$ b. else if v_2 is a nominal node or an ancestor of v_1, then $\mathsf{merge}(v_1, v_2)$ c. else $\mathsf{merge}(v_2, v_1)$
o-rule	if	there are two nodes v, v' with $\{a\} \in (\mathcal{L}(v) \cap \mathcal{L}(v'))$ and not $v \dot{\neq} v'$
	then	$\mathsf{merge}(v, v')$
NN-rule	if	1. $\leqslant n\, r.C \in \mathcal{L}(v)$, v is a nominal node, and there is a blockable r-neighbour v' of v such that $C \in \mathcal{L}(v')$ and v is a successor of v', 2. there is no m such that $1 \leq m \leq n$, $(\leqslant m\, r.C) \in \mathcal{L}(v)$, and there exist m nominal r-neighbours $v_1, \ldots, v_m$ of v with $C \in \mathcal{L}(v_i)$ and $v_i \dot{\neq} v_j$ for all $1 \leq i < j \leq m$
	then	1. guess m with $1 \leq m \leq n$ and $\mathcal{L}(v) = \mathcal{L}(v) \cup \{\leqslant m\, r.C\}$ 2. create m new nodes $v'_1, \ldots, v'_m$ with $\mathcal{L}(\langle v, v'_i \rangle) = \{r\}$, $\mathcal{L}(v'_i) = \{\top, C, \{a_i\}\}$ with each $a_i \in N_I$ new in G and $\mathcal{K}$, and $v'_i \dot{\neq} v'_j$ for $1 \leq i < j \leq m$.

Table 1: Tableau expansion rules for normalised $\mathcal{SROIQ}$ knowledge bases

During the expansion it is sometimes necessary to merge two nodes or to delete (prune) a part of the completion graph (Horrocks & Sattler, 2007). Roughly speaking, when a node w is merged into a node v, e.g., by an application of the $\leqslant$-rule, written $\mathsf{merge}(w, v)$, we add $\mathcal{L}(w)$ to $\mathcal{L}(v)$, "move" all the edges leading to w so that they lead to v and "move" all the edges leading from w to nominal nodes so that they lead from v to the same nominal nodes; we then remove w (and blockable sub-trees below w) from the completion graph, written $\mathsf{prune}(w)$, to prevent a further rule application on these nodes.

Note, in order to ensure termination of the tableau algorithm, it is in principle necessary to apply certain "crucial" rules with a higher priority. For example, the o-rule is applied with the highest priority and the NN-rule has to be applied before the $\leqslant$-rule. The priority of other rules is not relevant as long as they are applied with a lower priority than for these crucial rules.

3. Saturation Compatible with Tableau Algorithms

In this section, we describe a saturation method that is an adaptation of the completion-based procedure (Baader et al., 2005) such that it generates data structures that are compatible for further usage within a fully-fledged tableau algorithm. Roughly speaking, the saturation approximates completion graphs in a compressed form and, therefore, it directly allows the extraction and transfer of results from the saturation to the tableau algorithm. To be more precise, we ensure that the saturation generates nodes that are, similarly to the nodes in completion graphs, labelled with sets of concepts. The saturated labels can then be used to initialise the labels of new nodes in completion graphs or to block the processing of successors. Moreover, in some cases, it is directly possible to build a model from the data structures of the saturation, which makes the construction of completion graphs 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 knowledge bases that can easily be supported with an efficient algorithm (see Section 3.1), i.e., we simply ignore unsupported concept constructors or only process them partially, and afterwards (see Section 3.2), we dynamically detect which parts have not been completely handled by the saturation. 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.

For an easy integration into a highly optimised tableau procedure, the (usually simpler) saturation procedure is adapted to work on the same (normalised and preprocessed) knowledge base and data structures as the tableau algorithm (but, in principle, also the opposite direction would be possible). This enables, for example, a direct use of node labels from the saturation in the tableau algorithm. For this coupling technique, the use of a good absorption algorithm is crucial since the saturation only handles deterministic parts of the knowledge base.

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., by "reusing" nodes.

Definition 9 (Saturation Graph)**.** *A* saturation graph *for a knowledge base $\mathcal{K}$ 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 $\mathcal{L}(v) \subseteq \mathsf{fclos}(\mathcal{K})$ 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 $\mathcal{L}(\langle v, v' \rangle) \subseteq \mathsf{Rols}(\mathcal{K})$.*

Obviously, a saturation graph is a data structure that is very similar to a completion graph. A major difference is, however, the missing $\dot{\neq}$-relation, which can be omitted since the saturation is not designed to completely handle cardinality restrictions. Furthermore, each node in the saturation graph is the representative node for a specific concept, which allows for "reusing" nodes as successors. For example, instead of creating new successors for existential restrictions, we reuse the representative node for the existentially restricted concept as a successor.

Since nodes, edges, and labels are used as in completion graphs, we use the terms (r-)neighbour, (r-)successor, (r-)predecessor, ancestor, and descendant analogously. Please note, however, that all nodes in the saturation graph can have several predecessors due to the reuse of nodes, whereas in completion graphs, only the nominal nodes can have several predecessors.

We initialise the saturation graph with the representative nodes for those concepts that have to be saturated. For example, if the satisfiability of the concept C has to be tested, then we add the node v_C with the label $\mathcal{L}(v_C) = \{\top, C\}$ to the saturation graph. If we are later also interested in the saturation of a concept D, then we simply extend the existing saturation graph by v_D. For knowledge bases that contain nominals, we further initialise the saturation graph with a node $v_{\{a\}}$ with $\mathcal{L}(v_{\{a\}}) = \{\top, \{a\}\}$ for each nominal $\{a\}$ occurring in the knowledge base. The saturation then simply applies the rules depicted in Table 2 to the saturation graph.

Definition 10 (Saturation)**.** *Let $S = (V, E, \mathcal{L})$ be a saturation graph for a knowledge base $\mathcal{K}$, then the saturation of S exhaustively applies the rules of Table 2 to S. A saturation graph is called* fully saturated *if the rules are not further applicable. We use the function* $\mathsf{saturate}$ *to denote the saturation of a saturation graph S, i.e., $\mathsf{saturate}(S)$ returns S', where S' is the fully saturated extension of S.*

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 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 terminates since the rules are only applied when they can add new concepts or roles to node or edge labels. Moreover, a cycle detection such as blocking is not required, which makes rule application very fast. Note also that the $\forall$-rule propagates concepts only to the predecessors of a node. This restriction is necessary in order to allow the reuse of nodes for existentially restricted concepts.

To efficiently derive as many *sound* inferences as possible, some saturation rules in Table 2 only partially support more expressive features of $\mathcal{SROIQ}$. After a full saturation, we then check where the saturation graph is possibly *incomplete*. Although there are often

$\sqsubseteq_1$-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	$\mathcal{L}(v) = \mathcal{L}(v) \cup \{C\}$
$\sqsubseteq_2$-rule:	if	$\{A_1, A_2\} \subseteq \mathcal{L}(v)$, $A_1 \sqcap A_2 \sqsubseteq C \in \mathcal{K}$, and $C \notin \mathcal{L}(v)$
	then	$\mathcal{L}(v) = \mathcal{L}(v) \cup \{C\}$
$\sqcap$-rule:	if	$C_1 \sqcap C_2 \in \mathcal{L}(v)$ and $\{C_1, C_2\} \not\subseteq \mathcal{L}(v)$
	then	$\mathcal{L}(v) = \mathcal{L}(v) \cup \{C_1, C_2\}$
$\exists$-rule:	if	$\exists r.C \in \mathcal{L}(v)$ and $r \notin \mathcal{L}(\langle v, v_C \rangle)$
	then	$\mathcal{L}(\langle v, v_C \rangle) = \mathcal{L}(\langle v, v_C \rangle) \cup \{r\}$
$\forall$-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	$\mathcal{L}(v') = \mathcal{L}(v') \cup \{C\}$
$\sqcup$-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	$\mathcal{L}(v) = \mathcal{L}(v) \cup \{D\}$
$\geqslant$-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	$\mathcal{L}(\langle v, v_C \rangle) = \mathcal{L}(\langle v, v_C \rangle) \cup \{r\}$
Self-rule:	if	$\exists r.\mathsf{Self} \in \mathcal{L}(v)$ and v is not an r-successor of v
	then	$\mathcal{L}(\langle v, v \rangle) = \mathcal{L}(\langle v, v \rangle) \cup \{r\}$
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	$\mathcal{L}(v) = \mathcal{L}(v) \cup \{D\}$
$\bot$-rule:	if	$\bot \notin \mathcal{L}(v)$, and 1. $\{C, \mathsf{nnf}(\neg C)\} \subseteq \mathcal{L}(v)$, or 2. v is an r-successor of itself and $\{\neg\exists r.\mathsf{Self}, \neg\exists r^-.\mathsf{Self}\} \cap \mathcal{L}(v) \neq \emptyset$, or 3. v' is an r-successor of v with $\{a\} \in \mathcal{L}(v) \cap \mathcal{L}(v')$ and $\neg\exists r.\mathsf{Self} \in \mathcal{L}(v)$ or $\neg\exists r^-.\mathsf{Self} \in \mathcal{L}(v)$, or 4. $\{\geqslant n\, r.C, \leqslant m\, s.D\} \subseteq \mathcal{L}(v)$ with $n > m$, $r \sqsubseteq^* s$, and $D \in \mathcal{L}(v_C)$, or 5. $\geqslant n\, r.C \in \mathcal{L}(v)$ with $n > 1$ and $\{a\} \in \mathcal{L}(v_C)$, or 6. v' is an r-successor of v, $r \sqsubseteq^* s$, and $\mathsf{Disj}(r, s) \in \mathcal{K}$, or 7. v' is an r-successor of v and v'' is an s-successor of v and $\{a\} \in \mathcal{L}(v') \cap \mathcal{L}(v'')$ and $\mathsf{Disj}(r, s) \in \mathcal{K}$, or 8. there exists a successor node v' of v with $\bot \in \mathcal{L}(v')$, or 9. there exists a node $v_{\{a\}}$ with $\bot \in \mathcal{L}(v_{\{a\}})$
	then	$\mathcal{L}(v) = \mathcal{L}(v) \cup \{\bot\}$

Table 2: Saturation rules for (partially) handling $\mathcal{SROIQ}$ knowledge bases

several ways to integrate the support of more expressive concept constructors, we chose a simple one that only allows a partial saturation, but which can be implemented very efficiently. For instance, the $\sqcup$-rule adds only those concepts that are implied by both disjuncts. Hence, the addition of concepts by this rule is obviously sound, but its handling of disjunctions is often incomplete. For at-least cardinality restrictions, we build the edges to (possibly reused) successor nodes similarly to the $\exists$-rule. Thereby, the actual cardinality is ignored, which possibly causes incompleteness if also at-most cardinality restrictions for related super-roles are in the same label. In order to (partially) handle a nominal $\{a\}$ in the label of a node v, we use an o-rule that adds 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). As a consequence, the unsatisfiability of concepts of the form $\exists r.(A \sqcap \{a\}) \sqcap \exists r.(\neg A \sqcap \{a\})$ cannot be discovered. This simple implementation does, however, not require the repeated saturation of the same concepts extended by small influences from the nominals. While tractable and complete saturation algorithms with nominals are possible (Kazakov, Krötzsch, & Simančík, 2012), many ontologies use nominals in such a simple way that this o-rule is already sufficient (e.g., by using nominals only in concepts of the form $\exists r.\{a\}$).

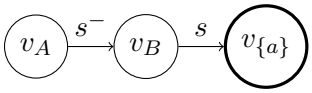
We further need an explicit $\bot$-rule that uses similar conditions as the ones for clashes in completion graphs for $\mathcal{SROIQ}$ (cf. Definition 7). The $\bot$-rule is used to handle several independent concepts in a one-pass manner within the same saturation graph and to distinguish nodes for unsatisfiable concepts from nodes that are (possibly) still satisfiable. The $\bot$-rule not only detects trivial reasons for unsatisfiability such as C and $\neg C$ in the label of a node (Condition 1), but also more involved cases. Violations regarding concepts of the form $\neg\exists r.\mathsf{Self}$ and $\neg\exists r^{-}.\mathsf{Self}$ are handled by Conditions 2 and 3. The former handles straightforward self-loops, while the latter handles cases that would lead to a loop if the saturation were to merge neighbouring nodes with the same nominal in their label. Conditions 4 and 5 handle problematic cases with cardinality restrictions. The actual cardinalities are ignored by the saturation, but it is clear that a clash would occur in a completion graph in the presence of conflicting at-least and at-most cardinalities (Condition 4) or for at-least number restrictions of the form $\geqslant n\, r.C$ with $n > 1$, where the node representing C contains a nominal. In the latter case, only one instance of C, the nominal, can exist in any model of the knowledge base. Conditions 6 and 7 handle problems with $\mathsf{Disj}(r, s)$ axioms. The former condition treats the more trivial case, where an r-successor is also an s-successor due to s being a super-role of r. The latter condition considers that the saturation does not merge nodes with the same nominal in their label, which would merge the labels of the edges from and to these nodes. Note that a node v_C can be both an r- and an s-successor due to node reuse in the saturation if the concepts $\exists r.C$ and $\exists s.C$ are in the label of the same node. This is, however, no problem even in the presence of $\mathsf{Disj}(r, s)$ axioms since this is not required for models of the knowledge base. The $\bot$-rule also propagates $\bot$ to ancestor nodes (Condition 8) and, in case $\bot$ occurs in the label of a nominal node, $\bot$ is propagated to every other node since the knowledge base is inconsistent (Condition 9).

It is in principle possible to detect also several other kinds of clashes for the incompletely handled parts in the saturation (e.g., for a concept C that has to be propagated to a successor node v with $\neg C \in \mathcal{L}(v)$), but the presented conditions of the $\bot$-rule, in combination with the detection of incompleteness (see Section 3.2), are already sufficient to identify all potential causes of unsatisfiability. Hence, we omit further clash conditions for ease of presentation.

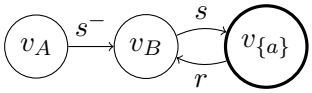
Example 1. *Let us assume that the TBox $\mathcal{T}_1^a$ contains the following axioms:*

$$A \sqsubseteq \exists s^{-}.B \qquad\qquad B \sqsubseteq \exists s.\{a\}$$

If we are interested in the satisfiability of the concept A, the saturation graph is initialised with the representative node for A, say v_A, with $\mathcal{L}(v_A) = \{\top, A\}$ and $v_{\{a\}}$ with $\mathcal{L}(v_{\{a\}}) = \{\top, \{a\}\}$ for the representation of the individual a. The $\sqsubseteq_1$-rule (cf. Table 2) is applicable for the first axiom and v_A, which results in the addition of $\exists s^{-}.B$ to $\mathcal{L}(v_A)$ for which the



$\mathcal{L}(v_A) = \{\top, A, \exists s^-.B\}$
$\mathcal{L}(v_B) = \{\top, B, \exists s.\{a\}\}$
$\mathcal{L}(v_{\{a\}}) = \{\top, \{a\}\}$



$\mathcal{L}(v_A) = \{\top, A, \exists s^-.B, B \sqcup \{a\}, C\}$
$\mathcal{L}(v_B) = \{\top, B, \exists s.\{a\}, C, \leqslant 1\, s.C\}$
$\mathcal{L}(v_{\{a\}}) = \{\top, \{a\}, C, \geqslant 2\, r.B\}$

Figure 1: Generated saturation graphs for testing the satisfiability of A w.r.t. $\mathcal{T}_1^a$ (left) and $\mathcal{T}_1^c$ (right) from Example 1

application of the $\exists$-rule generates the node v_B with $\mathcal{L}(v_B) = \{\top, B\}$ and an s^--labelled edge between v_A and v_B. Now, the $\sqsubseteq_1$-rule can be applied to unfold B in the label of v_B to $\exists s.\{a\}$ for which the $\exists$-rule is again applicable. Hence, we obtain the fully saturated saturation graph depicted on the left-hand side of Figure 1. Note that the saturation procedure starts the rule application with only the nodes v_A and $v_{\{a\}}$; other nodes, e.g., v_B, are created on demand.

Now let $\mathcal{T}_1^b = \mathcal{T}_1^a$ plus the axioms:

$$A \sqsubseteq B \sqcup \{a\} \qquad B \sqsubseteq C \qquad \{a\} \sqsubseteq C$$

The $\sqsubseteq_1$-rule extends $\mathcal{L}(v_A)$ with $B \sqcup \{a\}$ and $\mathcal{L}(v_B)$ as well as $\mathcal{L}(v_{\{a\}})$ with C. Now the $\sqcup$-rule is applicable and adds the concept C to the label of v_A because C is in the label of the representative nodes for both disjuncts of the disjunction $B \sqcup \{a\}$. Note that 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.

Finally, let $\mathcal{T}_1^c = \mathcal{T}_1^b$ plus the axioms:

$$B \sqsubseteq \leqslant 1\, s.C \qquad \{a\} \sqsubseteq \geqslant 2\, r.B$$

The $\sqsubseteq_1$-rule extends $\mathcal{L}(v_B)$ with $\leqslant 1\, s.C$ and $\mathcal{L}(v_{\{a\}})$ with $\geqslant 2\, r.B$. The latter addition triggers the $\geqslant$-rule, which adds an r-edge from $v_{\{a\}}$ to v_B such that the saturation graph depicted on the right-hand side of Figure 1 is obtained. Note that the saturation is inherently incomplete. For example, there is no saturation rule that handles the concept $\leqslant 1\, s.C$. The tableau rules would merge v_A and $v_{\{a\}}$ since v_B can have only one s-successor with C in its label, which possibly leads to conclusions that the saturation misses. We cover how such incomplete handling of nodes can be detected in the next section.

With a suitable absorption technique, the saturation is usually able to derive and add the majority of those concepts that would be added by the tableau algorithm for an equivalent node. This is especially the case for ontologies that primarily use features of the DL $\mathcal{EL}^{++}$ for which saturation-based procedures are particularly well-suited. 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 7.

3.2 Saturation Status Detection

Similarly to other saturation procedures, the presented method in Section 3.1 easily becomes incomplete for more expressive DLs. In order to nevertheless gain as much information as

possible from the saturation, we identify nodes for which the saturation was possibly incomplete. We call such nodes *critical*. In principle, such nodes can be detected by testing whether an actual tableau rule is still applicable. However, since we saturate some more expressive concept constructors partially, this approach is often too conservative. For example, for at-least cardinality restrictions of the form $\geqslant n\, r.C$ with $n > 1$, the saturation already creates or reuses a successor node with the concept C and, therefore, all consequences that are propagated back from this successor node are already considered. Nevertheless, the tableau expansion rule for this at-least cardinality restriction is still applicable, since we have not created n successors that are stated as pairwise different. This is, however, only relevant if there are some restrictions that limit the number of allowed r-successors with the concept C in their label. For the DL $\mathcal{SROIQ}$, such limitations are only possible with nominals and at-most cardinality restrictions. Therefore, it is sufficient to check for such limitations instead of testing whether the tableau expansion rules are applicable. Similar relaxations are also possible for other concept constructors.

We use the rules of Table 3 and 4 to detect the "saturation status" of a saturation graph, where incompletely handled nodes are identified and other information that is relevant for supporting the tableau algorithm is extracted. To be more precise, the rules are applied to a saturation graph and gather nodes in the sets $\mathcal{S}_{\mathsf{o}}$, $\mathcal{S}_{\not<}$, and $\mathcal{S}_{!}$, where $\mathcal{S}_{\mathsf{o}}$ represents nodes that "depend on nominals", $\mathcal{S}_{\not<}$ represents nodes with "tight at-most restrictions", and $\mathcal{S}_{!}$ represents "critical nodes" that are potentially not completely handled by the saturation. In order to specify the saturation status in more detail, we first define the number of merging candidates to facilitate the treatment of possibly incompletely handled at-most cardinality restrictions.

Definition 11 (Merging Candidates)**.** *Let $S = (V, E, \mathcal{L})$ be a saturation graph. For a role s and a concept D, the* number of merging candidates for a node $v \in V$ w.r.t. s and D, *written as the function $\sharp\mathsf{mcands}(v, s, D)$, is defined as $\sum_{\geqslant n\, r.C \in L} n$ with*

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

For an at-most cardinality restriction $\leqslant m\, s.D$ in the label of a node v, the merging candidates are those s-successors that have the concept D in their label. This is used by the $\mathcal{C}_{\not<}$-rule (see Table 3) to identify nodes with tight at-most restrictions, which is the case for a node v with an at-most cardinality restriction $\leqslant m\, s.D$ in its label if the number of merging candidates for v w.r.t. s and D is m, i.e., $m = \sharp\mathsf{mcands}(v, s, D)$. For such nodes, it is still not necessary to merge some of the merging candidates, but every additional candidate might require merging and, therefore, these nodes cannot be used arbitrarily.

The $\mathcal{C}_{\mathsf{o}}$-rule adds to the set $\mathcal{S}_{\mathsf{o}}$ all nodes that directly or indirectly depend on nominals, i.e., it identifies all nodes that directly have a nominal in their label or have a descendant node with a nominal in its label.

The rules of Table 4 are used to identify critical nodes for which the saturation procedure might be incomplete, i.e., these nodes are added to the set $\mathcal{S}_{!}$ as follows:

- The $\mathcal{C}_{\downarrow\forall}$- and $\mathcal{C}_{\sqcup}$-rule identify nodes as critical for which the $\forall$- or the $\sqcup$-rule of the tableau algorithm is applicable. Note, for the $\mathcal{C}_{\downarrow\forall}$-rule it is only necessary to check

$\mathcal{C}_{\not\leq}$-rule:	if	$v \notin \mathcal{S}_{\not\leq}$, $\leqslant m\, s.D \in \mathcal{L}(v)$, and $\sharp\mathsf{mcands}(v, s, D) = m$
	then	$\mathcal{S}_{\not\leq} = \mathcal{S}_{\not\leq} \cup \{v\}$
$\mathcal{C}_{\mathsf{o}}$-rule:	if	$v \notin \mathcal{S}_{\mathsf{o}}$ and either $\{a\} \in \mathcal{L}(v)$ or v has a successor node v' with $v' \in \mathcal{S}_{\mathsf{o}}$
	then	$\mathcal{S}_{\mathsf{o}} = \mathcal{S}_{\mathsf{o}} \cup \{v\}$

Table 3: Rules for detecting nodes with tight at-most restrictions and nodes with nominal dependency in the saturation graph

whether the concepts can be propagated to the successor nodes since the propagation to predecessors is ensured by the saturation procedure.

- The $\mathcal{C}_{\downarrow\leqslant}$-rule checks for every node v whether there is a potentially unsatisfied at-most cardinality restriction of the form $\leqslant m\, s.D$ in the label of v, i.e., $\sharp\mathsf{mcands}(v, s, D) > m$. Analogously to the ch-rule in the tableau algorithm, the $\mathcal{C}_{\downarrow\mathsf{ch}}$-rule identifies nodes as incompletely handled if they have s-successor nodes with neither D nor $\mathsf{nnf}(\neg D)$ in their label. In addition, we have to consider that the successors may have to be merged into a predecessor. Note that this has to be checked from the perspective of the predecessors due to the reuse of nodes. Therefore, we check with the $\mathcal{C}_{\uparrow\leqslant}$- and $\mathcal{C}_{\uparrow\mathsf{ch}}$-rule on a node v whether there exists an $\mathsf{inv}(s)$-successor node v' that has a tight at-most restriction for s, i.e., $\leqslant m\, s.D \in \mathcal{L}(v')$ and $\sharp\mathsf{mcands}(v, s, D) = m$. If v is a merging candidate, i.e., $D \in \mathcal{L}(v)$, or it would be necessary to apply the ch-rule for v, then we consider v critical. For example, if v has an s^{-}-successor v' with $\{\geqslant 3\, s.D, \leqslant 3\, s.D\} \subseteq \mathcal{L}(v')$, then the $\mathcal{C}_{\uparrow\leqslant}$-rule ($\mathcal{C}_{\uparrow\mathsf{ch}}$-rule) identifies v as critical if $D \in \mathcal{L}(v)$ ($\{D, \mathsf{nnf}(\neg D)\} \cap \mathcal{L}(v) \neq \emptyset$).

- We also need several rules for the detection of incompleteness related to nominals. First, we check with the $\mathcal{C}_{\mathsf{oo}}$-rule whether there are two nodes in the saturation graph that have the same nominal but different concepts in their label. In this case, the handling of the nominal is possibly incomplete since merging these nodes would also merge their labels. Note that if we saturate several independent concepts in the same saturation graph, then merging all nodes with the same nominal in their label is not always necessary. However, detecting when a merge is really required would involve a more expensive test. Since many ontologies for less expressive DLs use nominals in very simple ways, we opt for a simple and efficient solution. In addition, if a node v is nominal dependent, i.e., it has a descendant node with a nominal in its label, then arbitrary consequences could be propagated to it via other individuals/nominals. Hence, the $\mathcal{C}_{\mathsf{o}!}$-rule adds v to the set $\mathcal{S}_!$ of critical nodes if a representative node for an individual is not completely handled since we cannot guarantee that the saturation has derived all consequences for v. In contrast, the $\mathcal{C}_{\mathsf{o}\leqslant}$-rule checks for possible interactions between nominals and at-most cardinality restrictions. Such an interaction is handled by the NN-rule of the tableau algorithm, but cannot easily be handled by the saturation and we rather identify such nodes as critical.

- Finally, the $\mathcal{C}_{\uparrow}$-rule marks all predecessors of critical nodes as critical.

$\mathcal{C}_{\downarrow\forall}$-rule:	if	$v \notin \mathcal{S}_!$, $\forall r.C \in \mathcal{L}(v)$, there is an r-successor v' of v, and $C \notin \mathcal{L}(v')$
	then	$\mathcal{S}_! = \mathcal{S}_! \cup \{v\}$
$\mathcal{C}_{\sqcup}$-rule:	if	$v \notin \mathcal{S}_!$, $C \sqcup D \in \mathcal{L}(v)$, and $\{C, D\} \cap \mathcal{L}(v) = \emptyset$
	then	$\mathcal{S}_! = \mathcal{S}_! \cup \{v\}$
$\mathcal{C}_{\downarrow\leqslant}$-rule:	if	$v \notin \mathcal{S}_!$, $\leqslant m\, s.D \in \mathcal{L}(v)$, and $\sharp\mathsf{mcands}(v, s, D) > m$
	then	$\mathcal{S}_! = \mathcal{S}_! \cup \{v\}$
$\mathcal{C}_{\downarrow\mathsf{ch}}$-rule:	if	$v \notin \mathcal{S}_!$, $\leqslant m\, s.D \in \mathcal{L}(v)$, there is an s-successor v' of v, and $\mathcal{L}(v') \cap \{D, \mathsf{nnf}(\neg D)\} = \emptyset$
	then	$\mathcal{S}_! = \mathcal{S}_! \cup \{v\}$
$\mathcal{C}_{\uparrow\leqslant}$-rule:	if	$v \notin \mathcal{S}_!$, $D \in \mathcal{L}(v)$, v' is an $\mathsf{inv}(s)$-successor of v, $\leqslant m\, s.D \in \mathcal{L}(v')$, and $\sharp\mathsf{mcands}(v', s, D) = m$
	then	$\mathcal{S}_! = \mathcal{S}_! \cup \{v\}$
$\mathcal{C}_{\uparrow\mathsf{ch}}$-rule:	if	$v \notin \mathcal{S}_!$, v' is an $\mathsf{inv}(s)$-successor of v, $\leqslant m\, s.D \in \mathcal{L}(v')$, and $\mathcal{L}(v) \cap \{D, \mathsf{nnf}(\neg D)\} = \emptyset$
	then	$\mathcal{S}_! = \mathcal{S}_! \cup \{v\}$
$\mathcal{C}_{\mathsf{oo}}$-rule:	if	$v \notin \mathcal{S}_!$, $\{a\} \in \mathcal{L}(v)$, $\{a\} \in \mathcal{L}(v')$, and $\mathcal{L}(v) \not\subseteq \mathcal{L}(v')$
	then	$\mathcal{S}_! = \mathcal{S}_! \cup \{v\}$
$\mathcal{C}_{\mathsf{o}!}$-rule:	if	$v \notin \mathcal{S}_!$, $v \in \mathcal{S}_\mathsf{o}$, and there exist some node $v_{\{a\}} \in \mathcal{S}_!$
	then	$\mathcal{S}_! = \mathcal{S}_! \cup \{v\}$
$\mathcal{C}_{\mathsf{o}\leqslant}$-rule:	if	$v \notin \mathcal{S}_!$, v' is an $\mathsf{inv}(s)$-successor of v, $\{a\} \in \mathcal{L}(v')$, $\leqslant m\, s.D \in \mathcal{L}(v')$, and $\mathsf{nnf}(\neg D) \notin \mathcal{L}(v)$
	then	$\mathcal{S}_! = \mathcal{S}_! \cup \{v\}$
$\mathcal{C}_{\uparrow}$-rule:	if	$v \notin \mathcal{S}_!$, there is a successor v' of v, and $v' \in \mathcal{S}_!$
	then	$\mathcal{S}_! = \mathcal{S}_! \cup \{v\}$

Table 4: Rules for detecting incompleteness in the saturation graph

The sets $\mathcal{S}_\mathsf{o}$, $\mathcal{S}_{\not<}$, and $\mathcal{S}_!$ are now used to define the saturation status of a saturation graph as follows:

Definition 12 (Saturation Status)**.** *The* saturation status $\mathcal{S}$ *of a saturation graph* $S = (V, E, \mathcal{L})$ *is defined as the tuple* $(\mathcal{S}_\mathsf{o}, \mathcal{S}_{\not<}, \mathcal{S}_!)$*. We use* status *as the function that creates* $\mathcal{S}$ *from* S *by the exhaustive application of the rules in Table 3 and 4. A node* $v \in V$ *is* critical *if* $v \in \mathcal{S}_!$*,* v *is* nominal dependent *if* $v \in \mathcal{S}_\mathsf{o}$*, and* v *has* tight at-most restrictions *if* $v \in \mathcal{S}_{\not<}$*. We call* v clashed *if* $\bot \in \mathcal{L}(v)$*.*

Note that a concept C is unsatisfiable if its representative node v_C is clashed. The satisfiability of C can, however, only be guaranteed if v_C is not critical and the knowledge base is consistent. Consistency is required, because a concept is satisfiable only if the knowledge base is consistent, which can only be determined by the saturation if no nominal node is critical.

Of course, if the satisfiability/completeness of saturated concepts is considered in the context of arbitrary other concepts that are not handled by the saturation (e.g., in a completion graph constructed by the tableau algorithm), then nominal dependency becomes more relevant. In particular, if new consequences are propagated to nominal nodes, then

the nominal dependent nodes in the saturation graph could be affected and they have to be considered as incompletely handled. Hence, also the satisfiability hinges on the status of the nominal nodes a node depends on.

A problem in practice is that a critical node for a nominal also makes all nominal dependent nodes critical. Hence, we can easily get many critical nodes for ontologies that use nominals if the saturation cannot completely handle all individuals. However, we can improve the saturation graph after the initial consistency check with the tableau algorithm (see Section 5 for details) by replacing the node labels of critical nominal nodes in the saturation graph with the ones from the obtained completion graph. Although we have to distinguish deterministically and non-deterministically derived concepts in these labels, we know that they correspond to a clash-free and fully expanded completion graph and, therefore, we can consider them as not critical.

Example 2. *Consider again the TBoxes from Example 1. We start with the fully saturated saturation graph for $\mathcal{T}_1^a$ (left-hand side of Figure 1). Only the $\mathcal{C}_{\mathsf{o}}$-rule from Table 3 is applicable, which identifies all nodes as nominal dependent and adds them (iteratively) to $\mathcal{S}_{\mathsf{o}}$, but all nodes are completely handled by the saturation and no node is critical or has tight at-most restrictions.*

The situation changes for the extension $\mathcal{T}_1^b$ of $\mathcal{T}_1^a$. Since $B \sqcup \{a\} \in \mathcal{L}(v_A)$, but neither disjunct is, v_A is identified as critical ($v_A \in \mathcal{S}_!$) by the $\mathcal{C}_\sqcup$-rule from Table 4. The other nodes are, however, still completely handled by the saturation.

Finally, consider the extension $\mathcal{T}_1^c$ of $\mathcal{T}_1^b$ (right-hand side of Figure 1). The $\mathcal{C}_{\not\leqslant}$-rule from Table 3 now adds v_B to $\mathcal{S}_{\not\leqslant}$ since the number of merging candidates for v_B w.r.t. s and C is 1, i.e., $\sharp\mathsf{mcands}(v_B, s, C) = 1$. This is used to identify nodes as critical that use v_B as an s^--successor such that merging with an s-successor of v_B is potentially required. In particular, the concept $\exists s^-.B \in \mathcal{L}(v_A)$ is now problematic because it connects v_B with v_A via the role s, $\leqslant 1\, s.C \in \mathcal{L}(v_B)$, and $\sharp\mathsf{mcands}(v_B, s, C) = 1$. Hence, if v_A were not already identified as critical due to the incompletely handled disjunction, the $\mathcal{C}_{\uparrow\leqslant}$-rule would add v_A to $\mathcal{S}_!$. Note, however, that v_B and $v_{\{a\}}$ are still completely handled by the saturation.

3.3 Correctness

It is straightforward to see that the saturation rules of Table 2 only produce sound inferences. In particular, the saturation rules add only those concepts to a label of a node which are also added by the tableau algorithm for an equivalently labelled node in the completion graph. The termination of the saturation rules is ensured since the number of nodes and edges as well as the size of the labels is bounded by the number of concepts, roles, and size of the closure of concepts in the knowledge base. Furthermore, the rules are only applied if they add new facts in the saturation graph. Analogously, the application of the rules of Table 3 and 4 for the generation of a saturation status is terminating, because each rule application adds the node to a corresponding set (either $\mathcal{S}_{\mathsf{o}}$, $\mathcal{S}_{\not\leqslant}$, or $\mathcal{S}_!$) and the rules are only applicable if the node does not already belong to the corresponding set.

It remains to show the completeness, i.e., we show that if a node v_D and the nodes representing individuals are neither critical nor clashed, we can build a model of the knowledge base in which the extension of D is non-empty. Note that a direct transformation of the saturation graph into a completion graph is not possible since the reuse of nodes in the

saturation graph possibly causes problems with certain features of $\mathcal{SROIQ}$. For example, if two roles r and s are stated disjoint and s is not a super role of r, then the saturation graph can contain a node that is an r- and an s-successor of another node. However, in principle, it would be possible to rebuild a completion graph by recursively creating corresponding successor nodes from the used nodes in the saturation graph until we would reach nominal nodes or the nodes would be blocked.

Given a fully saturated saturation graph, where all nodes representing individuals are neither critical nor clashed, we show completeness for a non-critical node v_D by providing an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ that is a model of the knowledge base with a non-empty extension of D, i.e., $D^{\mathcal{I}} \neq \emptyset$ and the interpretation witnesses the satisfiability of D. For ease of presentation, we assume that existentially quantified concepts of the form $\exists r.C$ are equivalently expressed as $\geqslant 1\, r.C$. This is w.l.o.g., since normalised knowledge bases contain only simple roles.

Since the occurrence of a nominal $\{a\}$ in the label of a node v_C means that the node v_C represents the same element as $v_{\{a\}}$ in $\Delta^{\mathcal{I}}$, we only need one representative element, which we ensure by defining a suitable equivalence relation over the nodes in the saturation graph.

Definition 13 (Canonical Saturation Model)**.** *For a saturation graph $S = (V, E, \mathcal{L})$, let $\approx$ be the following relation: $\{(v_C, v_C) \mid v_C \in V\} \cup \{(v_C, v_{\{a\}}) \mid v_C \in V, \{a\} \in \mathcal{L}(v_C), \mathcal{L}(v_{\{a\}}) = \mathcal{L}(v_C)\}$ and let $\overset{*}{\approx}$ be the transitive, reflexive, and symmetric closure of $\approx$. Since the relation $\overset{*}{\approx}$ is an equivalence relation over the nodes in V, we use $v_{[C]}$, for $v_C \in V$, to denote the equivalence class of v_C by $\overset{*}{\approx}$. We use the relation $\overset{*}{\approx}$ to (recursively) define the elements of $\Delta^{\mathcal{I}}$. We first define the set $\mathsf{Nom}(S) = \{v_{[\{a\}]} \mid a \in N_I\}$ of nodes with nominals in their labels. Further non-nominal elements of $\Delta^{\mathcal{I}}$ are then obtained by "unravelling" parts of the saturation graph into paths as usual (Horrocks & Sattler, 2007). We set*

$$\begin{aligned}\mathsf{Paths}_S(D) = \ & \{v_{[D]}\} \cup \mathsf{Nom}(S) \cup \\ & \{p \cdot v^i_{[C]} \mid p \in \mathsf{Paths}_S(D), \geqslant n\, r.C \in \mathcal{L}(v), v \in \mathsf{tail}(p), v_{[C]} \notin \mathsf{Nom}(S), 1 \leq i \leq n\},\end{aligned}$$

where $\cdot$ denotes concatenation and $\mathsf{tail}(v_{[C]}) = \mathsf{tail}(p \cdot v^i_{[C]}) = v_{[C]}$.

We can now define the interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ as follows:

$$\Delta^{\mathcal{I}} = \mathsf{Paths}_S(D),$$

and, for each $a \in N_I$, we set $\cdot^{\mathcal{I}}$ to

$$a^{\mathcal{I}} = v_{[\{a\}]},$$

for each $A \in N_C$ to

$$A^{\mathcal{I}} = \{p \mid v \in \mathsf{tail}(p) \text{ and } A \in \mathcal{L}(v)\},$$

and for each $r \in N_R$ *to*

$$\begin{aligned} r^{\mathcal{I}} = \{\langle p, q\rangle \in \mathsf{Paths}_S(D) \times \mathsf{Paths}_S(D) \mid\ & q = p \cdot v^i_{[C]} \text{ and } \geqslant n\, s.C \in \mathcal{L}(v) \\ & \text{for a } v \in \mathsf{tail}(p) \text{ with } i \leq n \text{ and } s \sqsubseteq^* r\} \cup \\ \{\langle q, p\rangle \in \mathsf{Paths}_S(D) \times \mathsf{Paths}_S(D) \mid\ & q = p \cdot v^i_{[C]} \text{ and } \geqslant n\, \mathsf{inv}(s).C \in \mathcal{L}(v) \\ & \text{for a } v \in \mathsf{tail}(p) \text{ with } i \leq n \text{ and } s \sqsubseteq^* r\} \cup \\ \{\langle p, x\rangle \in \mathsf{Paths}_S(D) \times \mathsf{Nom}(S) \mid\ & \text{there is a } v \in x \text{ and a } v' \in \mathsf{tail}(p) \\ & \text{such that } v \text{ is an } r\text{-successor of } v'\} \cup \\ \{\langle x, p\rangle \in \mathsf{Nom}(S) \times \mathsf{Paths}_S(D) \mid\ & \text{there is a } v \in x \text{ and a } v' \in \mathsf{tail}(p) \\ & \text{such that } v \text{ is an } \mathsf{inv}(r)\text{-successor of } v'\} \cup \\ \{\langle p, p\rangle \in \mathsf{Paths}_S(D) \times \mathsf{Paths}_S(D) \mid\ & \text{there is a } v \in \mathsf{tail}(p) \\ & \text{such that } v \text{ is an } r\text{-neighbour of itself }\} \end{aligned}$$

Note that the domain elements are (paths of) equivalent classes and to ensure that the extension of a role r correctly contains all edges derived by the saturation, we use all nodes of an equivalent class for the construction of $r^{\mathcal{I}}$.

In order to show that $\mathcal{I} \models \mathcal{K}$, we first show that, for every $p \in \Delta^{\mathcal{I}}$, it holds that $p \in C^{\mathcal{I}}$ if $C \in \mathcal{L}(v)$ with $v \in \mathsf{tail}(p)$.

Lemma 1. *Let $S = (V, E, \mathcal{L})$ be a fully saturated saturation graph for a concept D w.r.t. $\mathcal{K}$ and v_D as well as the nodes representing individuals are neither critical nor clashed. Furthermore, let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ denote an interpretation constructed as described in Definition 13. For each $p \in \Delta^{\mathcal{I}}$ it holds that $p \in C^{\mathcal{I}}$ if $C \in \mathcal{L}(v)$ with $v \in \mathsf{tail}(p)$.*

Proof 1. *We observe that all nodes involved in the construction of the interpretation can neither be critical nor clashed. In particular, if a node $v \in \mathsf{tail}(p)$ for a $p \in \Delta^{\mathcal{I}}$ were clashed, then v would be a descendant node of $v_{[D]}$ or of a node representing an individual and, hence, they would be identified as clashed with the $\bot$-rule, which contradicts our assumption that these nodes are not clashed. Analogously, if v were critical, then $v_{[D]}$ or a node representing an individual would be identified as critical by the $\mathcal{C}_\uparrow$-rule (which is also contradictory to our assumption). Hence, for considering the different types of concepts for proving the lemma in the following, it is safe to assume that all nodes used for the construction of the interpretation are neither clashed nor critical and, hence, $\bot^{\mathcal{I}} = \emptyset$.*

The base case for $C = A$ with $A \in \mathcal{L}(v)$ and $v \in \mathsf{tail}(p)$ trivially holds for all $p \in \Delta^{\mathcal{I}}$ by the definition of $\Delta^{\mathcal{I}}$ and $\cdot^{\mathcal{I}}$, i.e., $p \in A^{\mathcal{I}}$ if $A \in \mathcal{L}(v)$ with $v \in \mathsf{tail}(p)$. Also note that $\Delta^{\mathcal{I}} = \top^{\mathcal{I}}$ due to the definition of the saturation algorithm (in particular due to the initialisation of nodes) and due to the fact that we never remove concepts from labels in a saturation graph. Other base cases hold for all elements $p \in \Delta^{\mathcal{I}}$ with $v \in \mathsf{tail}(p)$ and $C \in \mathcal{L}(v)$ as follows:

- *For $C = \{a\}$, we observe that $p = v_{[\{a\}]}$ due to the use of equivalence classes, the definition of $\approx^*$, $\Delta^{\mathcal{I}}$, $\cdot^{\mathcal{I}}$, and the fact that the $\mathcal{C}_{\mathsf{oo}}$-rule cannot identify v as critical by assumption. Hence, $p \in C^{\mathcal{I}}$.*

- *For $C = \neg B$, we observe that $p \notin B^{\mathcal{I}}$ due to the definition of $\mathcal{I}$ and the fact that the used nodes are not clashed in the saturation graph. Hence, $p \in C^{\mathcal{I}}$.*

- *For $C = \exists r.\mathsf{Self}$, we observe that the node v is an r-successor of itself due to the application of the Self-rule. By the definition of $\cdot^{\mathcal{I}}$, we have $\langle p, p\rangle \in r^{\mathcal{I}}$. Hence, $p \in C^{\mathcal{I}}$.*

- *For $C = \neg\exists r.\mathsf{Self}$, we observe that the nodes in the saturation graph are not clashed. Hence, we can exclude loops that are caused by the last part of the definition of $\cdot^{\mathcal{I}}$ since any such node would be clashed due to Condition 2 of the $\bot$-rule. For nominal nodes we observe that nodes of V with the same nominal in their label are represented by one element in $\Delta^{\mathcal{I}}$. This possibly introduces loops for neighbouring nominal nodes in the saturation graph. This is, however, excluded by Condition 3 of the $\bot$-rule. Hence, we have $\langle p, p\rangle \notin r^{\mathcal{I}}$ and, therefore, $p \in C^{\mathcal{I}}$.*

The complex cases hold for all elements $p \in \Delta^{\mathcal{I}}$ with $v \in \mathsf{tail}(p)$ and $C \in \mathcal{L}(v)$ by induction as follows:

- *For $C = C_1 \sqcap C_2$, the application of the $\sqcap$-rule ensures that $\mathcal{L}(v) \supseteq \{C_1, C_2\}$. By induction, we have $p \in C_1^{\mathcal{I}}$ and $p \in C_2^{\mathcal{I}}$. Hence, $p \in C^{\mathcal{I}}$.*

- *For $C = C_1 \sqcup C_2$, we observe that there must be a concept $C' \in \mathcal{L}(v)$ with $C' \in \{C_1, C_2\}$ since the $\mathcal{C}_{\sqcup}$-rule would otherwise have identified the node $v \in \mathsf{tail}(p)$ as critical, which contradicts our assumption. Hence, by induction, we have $p \in C'^{\mathcal{I}}$ and, as a consequence, $p \in C^{\mathcal{I}}$.*

- *For $C = \geqslant n\, r.C'$, we observe that the saturation algorithm creates and saturates the node $v_{C'}$ as r-successor of v for $n \geq 1$ and we consider two cases: First, for $v_{[C']} \in \mathsf{Nom}(S)$, we have that $n = 1$ since v would be clashed for $n > 1$ due to Condition 5 of the $\bot$-rule, which contradicts our assumption. By the definition of $\mathsf{Paths}_S(D)$, $\Delta^{\mathcal{I}}$, and $\cdot^{\mathcal{I}}$, there exists the element q with $q \in \mathsf{Nom}(S)$ such that $\langle p, q\rangle \in r^{\mathcal{I}}$. Furthermore, by induction, we have $q \in C'^{\mathcal{I}}$ and, consequently, $p \in C^{\mathcal{I}}$. Second, for $v^i_{[C']} \notin \mathsf{Nom}(S)$, by the construction of $\mathsf{Paths}_S(D)$, $p \cdot v^1_{[C']}, \ldots, p \cdot v^n_{[C']} \in \mathsf{Paths}_S(D)$ and, by definition of $\cdot^{\mathcal{I}}$, these elements are n r-successors of p. Finally, by induction, we have $p \cdot v^1_{[C']}, \ldots, p \cdot v^n_{[C']} \in C'^{\mathcal{I}}$ and, consequently, $p \in C^{\mathcal{I}}$.*

- *For $C = \forall r.C'$, we observe that the application of the $\forall$-rule guarantees that all $\mathsf{inv}(r)$-predecessors have C' in their label. Furthermore, also all r-successors have C' in their label, otherwise the $\mathcal{C}_{\downarrow\forall}$-rule would have identified v as critical, which is contradictory to our assumption. Hence, by definition of $\mathsf{Paths}_S(D)$, $\Delta^{\mathcal{I}}$, $\cdot^{\mathcal{I}}$, and by induction, it holds for every r-neighbour element q that $q \in C'^{\mathcal{I}}$ and, consequently, $p \in C^{\mathcal{I}}$.*

- *For $C = \leqslant n\, r.C'$, the $\mathcal{C}_{\downarrow\leqslant}$-rule guarantees that every node $v \in p$ has at most n merging candidates, i.e., at most n r-successor nodes with C' in their labels, otherwise the node would be critical, which contradicts our assumption. Analogously, the $\mathcal{C}_{\downarrow\mathsf{ch}}$-rule guarantees that every r-successor of v has either C' or $\mathsf{nnf}(\neg C')$ in its label. For $p = v_{[D]}$ with $p \notin \mathsf{Nom}(S)$, we observe that p does not have any predecessors and,*

by definition of $\mathsf{Paths}_S(D)$, $\Delta^{\mathcal{I}}$, *and* $\cdot^{\mathcal{I}}$, *and by induction, we have at most* n r*-neighbour elements* $q_1, \ldots, q_n$ *with* $q_1, \ldots, q_n \in C'^{\mathcal{I}}$ *since for every other* r*-neighbour element* q *it holds that* $q \in \mathsf{nnf}(\neg C')^{\mathcal{I}}$ *and, therefore,* $q \notin C'^{\mathcal{I}}$. *Hence,* $p \in C^{\mathcal{I}}$ *for* $p = v_{[D]}$. *For* $p \in \mathsf{Nom}(S)$, *it holds, for every* $\mathsf{inv}(r)$*-predecessor element* q *of* p, *that* $q \notin C'^{\mathcal{I}}$ *due to induction, the definition of* $\mathsf{Paths}_S(D)$, $\Delta^{\mathcal{I}}$, *and* $\cdot^{\mathcal{I}}$, *and the* $\mathcal{C}_{\mathsf{o}\leqslant}$*-rule (otherwise the node would be identified as critical). Hence,* $p \in C^{\mathcal{I}}$ *for* $p \in \mathsf{Nom}(S)$. *For* $p \notin \{v_{[D]}\} \cup \mathsf{Nom}(S)$, *we have* $p = q \cdot v^i_{[D']}$. *Moreover, the* $\mathcal{C}_{\uparrow\mathsf{ch}}$*-rule guarantees that for* $w \in \mathsf{tail}(q)$ *either* C' *or* $\mathsf{nnf}(\neg C')$ *is in its label. By induction and the definition of* $\mathsf{Paths}_S(D)$, $\Delta^{\mathcal{I}}$, *and* $\cdot^{\mathcal{I}}$, *we have either* $q \in C'^{\mathcal{I}}$ *or* $q \in \mathsf{nnf}(\neg C')^{\mathcal{I}}$. *We consider both cases: First, if* $q \notin C'^{\mathcal{I}}$ *because of* $w \in \mathsf{tail}(q)$ *with* $\mathsf{nnf}(\neg C') \in \mathcal{L}(w)$, *then we can argue analogously to the case where* $p = v_{[D]}$. *Second, if* $q \in C'^{\mathcal{I}}$ *because of* $w \in \mathsf{tail}(q)$ *with* $C' \in \mathcal{L}(w)$, *then it is guaranteed by the* $\mathcal{C}_{\uparrow\leqslant}$*-rule that* v *has at most* $n-1$ r*-successors with* C *in their labels and, by definition of* $\mathsf{Paths}_S(D)$, $\Delta^{\mathcal{I}}$, *and* $\cdot^{\mathcal{I}}$, *we have at most* n r*-neighbour elements* $q_1, \ldots, q_n$ *of* p *for which it holds by induction that* $q_1, \ldots, q_n \in C'^{\mathcal{I}}$. *Consequently,* $p \in C^{\mathcal{I}}$.

By using Lemma 1, we can now show the completeness, i.e., we can show that the constructed interpretation satisfies all axioms in the knowledge base:

Lemma 2 (Completeness)**.** *Let* $S = (V, E, \mathcal{L})$ *be a fully saturated saturation graph for a concept* D *w.r.t.* $\mathcal{K}$ *and* v_D *as well as the nodes representing individuals are neither critical nor clashed, then there exists an interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *such that* $\mathcal{I} \models \mathcal{K}$ *and* $D^{\mathcal{I}} \neq \emptyset$.

Proof 2. *We assume that the interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *is built from* S *as in Definition 13. Note that due to the definition of the saturation algorithm, there is a node* $v_D \in V$ *with* $D \in \mathcal{L}(v_D)$, *by the definition of* $\Delta^{\mathcal{I}}$, $v_{[D]} \in \Delta^{\mathcal{I}}$ *and, by the definition of* tail *and* $\cdot^{\mathcal{I}}$, $v_{[D]} \in D^{\mathcal{I}}$. *Hence,* $D^{\mathcal{I}} \neq \emptyset$. *We can now observe that* $\mathcal{I}$ *satisfies every axiom* α *of the normalised knowledge base* $\mathcal{K}$ *(cf. Section 2.2) as follows:*

- *For* $\alpha = H \sqsubseteq C$ *with* $H = \{a\}$, $H = A$, *or* $H = \top$, *we observe that, for every* $p \in \Delta^{\mathcal{I}}$ *with* $v \in \mathsf{tail}(p)$, *we have* $H^{\mathcal{I}}$ *only if* $H \in \mathcal{L}(v)$ *due to the definition of* $\overset{*}{\approx}$, $\mathsf{Paths}_S(D)$, $\Delta^{\mathcal{I}}$, *and* $\cdot^{\mathcal{I}}$. *Due to the applications of the* $\sqsubseteq_1$*-rule, we have* $C \in \mathcal{L}(v)$ *if* $H \in \mathcal{L}(v)$, *and, by Lemma 1, we have* $p \in C^{\mathcal{I}}$ *if* $p \in H^{\mathcal{I}}$. *Hence,* $\mathcal{I} \models \alpha$.

- *For* $\alpha = A_1 \sqcap A_2 \sqsubseteq C$, *we analogously observe that, for every* $p \in \Delta^{\mathcal{I}}$ *with* $v \in \mathsf{tail}(p)$, *we have* $(A_1 \sqcap A_2)^{\mathcal{I}}$ *only if* $\{A_1, A_2\} \subseteq \mathcal{L}(v)$ *due to the definition of* $\overset{*}{\approx}$, $\mathsf{Paths}_S(D)$, $\Delta^{\mathcal{I}}$, *and* $\cdot^{\mathcal{I}}$. *Due to the applications of the* $\sqsubseteq_2$*-rule, we have* $C \in \mathcal{L}(v)$ *if* $\{A_1, A_2\} \subseteq \mathcal{L}(v)$, *and, by Lemma 1, we have* $p \in C^{\mathcal{I}}$ *if* $p \in H^{\mathcal{I}}$. *Hence,* $\mathcal{I} \models \alpha$.

- *For* $\alpha = r \sqsubseteq s$, *we observe that* $\mathcal{I} \models \alpha$ *due to the definition of successors/predecessors and by the definition of* $\mathsf{Paths}_S(D)$, $\Delta^{\mathcal{I}}$, *and* $\cdot^{\mathcal{I}}$.

- *For* $\alpha = \mathsf{Disj}(r, s)$, *we assume for a contradiction that there are the elements* $p, q \in \Delta^{\mathcal{I}}$ *such that* $\langle p, q\rangle \in r^{\mathcal{I}} \cap s^{\mathcal{I}}$. *By definition of a saturation graph and* $\cdot^{\mathcal{I}}$ *from the edges in* S, *either* $q \in \mathsf{Nom}(S)$ *or* $r \sqsubseteq^* s$. *However, in both cases* $v \in \mathsf{tail}(p)$ *is clashed due to Condition 6 and Condition 7 of the* $\bot$*-rule, respectively. Since it contradictory to our assumption that* v *is clashed, we have* $\mathcal{I} \models \alpha$.

Since $\mathcal{I}$ satisfies, for all elements of $\Delta^{\mathcal{I}}$, every axiom α of $\mathcal{K}$, $\mathcal{I} \models \mathcal{K}$. □

From the language features that are completely supported by the presented saturation algorithm and from the fact that $\exists r.C$ concepts on the left-hand side of GCIs can be transformed to universal restrictions on the right-hand side (with only propagations to predecessors if inverse roles are not used), one can observe that completeness of the presented saturation algorithm is guaranteed for at least $\mathcal{ELH}$ knowledge bases.

4. Supporting Tableau Algorithms

In this section, we present a range of optimisations to directly and indirectly support reasoning with tableau algorithms for the DL $\mathcal{SROIQ}$. As already mentioned, reasoning systems for more expressive DLs are usually very complex and they integrate many sophisticated optimisations which are necessary to make reasoning for many real-world ontologies practicable. As a consequence, it is important for the development of new optimisations to consider the interaction with already existing techniques. For example, a very important and well-known optimisation technique is dependency directed backtracking which allows for only evaluating "relevant" non-deterministic alternatives with the tableau algorithm. A typical realisation of dependency directed backtracking is backjumping where every fact that is added to the completion graph is labelled with those non-deterministic branches on which the fact depends (Baader et al., 2007; Tsarkov, Horrocks, & Patel-Schneider, 2007). If a clash is discovered, then we can jump back to the last non-deterministic decision that is referenced by the clashed facts in the completion graph and, consequently, we do not have to evaluate non-deterministic alternatives for which it is clear that they would result in the same clashes. Hence, new optimisation techniques that manipulate completion graphs must obviously also add these dependencies correctly, otherwise dependency directed backtracking cannot be completely supported in the presence of these optimisations.

The optimisation techniques that we present in this section are fully compatible with dependency directed backtracking and, to the best of our knowledge, they also do not negatively influence other well-known optimisations. Moreover, since the saturation optimisations allow for doing a lot of reasoning work very efficiently, they often reduce the effort for other optimisation techniques. For example, these optimisations directly perform many simple expansions in the completion graph and, therefore, the effort for conventional caching methods is often reduced. In particular, tableau-based reasoning systems often cache satisfiable node labels in order to block the expansion of successors of identically labelled nodes in subsequent completion graphs. If we can "reuse" the labels of non-critical and non-clashed nodes from the saturation graph, then we directly know that a further processing in the completion graph is not required for them, even without checking whether corresponding node labels are in the satisfiability cache.

4.1 Transfer of Saturation Results to Completion Graphs

Since the presented saturation method uses compatible data structures, we can directly transfer the saturation results into completion graphs. This improves the tableau algorithm by a faster clash detection and optimises the construction of a completion graph. For example, we can directly use unsatisfiability information that is detected by the $\bot$-rule in

the saturation. In particular, if the application of a tableau expansion rule adds a concept C to the completion graph, then we can check the saturation status of v_C and, in case it is clashed, we can immediately initiate the backtracking with the dependencies from the unsatisfiable concept C in the completion graph. Analogously, we can utilise other derived consequences form the saturation. For instance, if an expansion rule adds a concept C to a label of a node in the completion graph, then we can add all concepts of $\mathcal{L}(v_C)$ to the same label. Of course, in order to further support dependency directed backtracking, we also have to add the correct dependencies. However, since all concepts of $\mathcal{L}(v_C)$ are deterministic consequences of C, we can simply use that D deterministically depends on C for every additionally added concept $D \in \mathcal{L}(v_C)$.

As a nice side effect, the addition of derived concepts from the saturation improves the backtracking and processing of disjunctions. Basically, the $\sqcup$-rule from the saturation extracts shared (super-)concepts from all disjuncts of a disjunction. For example, for the disjunction $A_1 \sqcup A_2$ and the axioms $A_1 \sqsubseteq B$ and $A_2 \sqsubseteq B$, we derive with the saturation that $\mathcal{L}(v_{A_1 \sqcup\ A_2}) \supseteq \{A_1 \sqcup A_2, B\}$, i.e., B is a super-concept of both disjuncts and we can add it as a deterministic consequence of the disjunction $A_1 \sqcup A_2$. Although we still have to process the disjunction, we can add some of the consequences (e.g., B) deterministically. Hence, backtracking does not identify the processing of alternatives of a disjunction as relevant if only such deterministic consequences are involved in the clash.

The transfer of derived consequences to directly add as many consequences as possible is helpful in several ways. First, the application of expansion rules from the tableau algorithms might become unnecessary. For example, if a disjunct of a disjunction has already been added, then it is not necessary to apply the $\sqcup$-rule. Second, if specific concepts are in the label of a node, then, at least for some expansion rules of the tableau algorithm, optimised rule applications are possible. For instance, if the concepts $\exists r.C$, $\exists r.D$, and $\leqslant 1\, r.\top$ are in the label of the same node, then the second application of the $\exists$-rule by the tableau algorithm can directly add the existentially restricted concept to the already present r-successor instead of creating a new one that has to be merged afterwards. Third, if concepts are propagated back to ancestor nodes, then it is necessary to again check whether one of the modified ancestor nodes is blocked before rules on descendant nodes can be applied. Due to the transfer of derived consequences, many of the concepts that are propagated back from successors are already added and, therefore, the amount of blocking tests is significantly reduced. Last but not least, the transfer of derived consequences allows for blocking much earlier. Blocking of a node v is usually only possible if a node could be replaced by another non-blocked node from the completion graph that does not influence any ancestor of v. A simple blocking condition that guarantees completeness for more expressive DLs is pairwise blocking. However, pairwise blocking can be refined to achieve more precise blocking conditions that possibly allow for blocking earlier (Horrocks & Sattler, 2001). Since many of the concepts that are propagated back from successors are added by the transfer of derived consequences from the saturation, it is likely that the creation and processing of new successor nodes does not influence ancestor nodes. As a result, it might be possible to block nodes even without the creation and processing of many successors.

Besides the transfer of derived consequences, it is in some cases also possible to directly block the processing of successor nodes in the completion graph. For this, the node in the completion graph, say v, has to be labelled with the same concepts as a node v' in the

saturation graph and v' must neither be clashed, critical, nor nominal dependent. If there exists such a v', then the processing of the successors of v can be blocked since v could be expanded in the same way as v' in the saturation graph. Obviously, we have to enforce that v' is not nominal dependent, because a dependent nominal could be influenced in the completion graph such that new consequences are propagated back to v and this would not be considered if the processing of successor nodes is blocked. Furthermore, it is indeed necessary to create the successors before blocking their processing, because they may have to be merged into the ancestor node. However, if the saturation node v' does not have a tight at-most restriction, i.e., for each at-most cardinality restriction $\leqslant m\, r.C \in \mathcal{L}(v')$, v' has at most $m-1$ r-successors that have not $\mathsf{nnf}(\neg C)$ in their label, then also the creation of successor nodes can be blocked, because every at-most cardinality restriction in the label of the node allows for at least one additional neighbour before some nodes have to be merged. Since nodes can easily have a large number of successors (e.g., due to at-least cardinality restrictions with big cardinalities), blocking the creation of new successors can be a significant improvement in terms of memory consumption and building time of the completion graph. Of course, if new concepts are propagated to v such that the label of v differs from v', then the blocking becomes invalid and the processing of the successors has to be reactivated or we have to find another compatible blocker node.

4.2 Subsumer Extraction

For tableau-based reasoning systems, many higher level reasoning task are often reduced to consistency checking. For example, a very naive classification algorithm tests the satisfiability of all classes and then checks the pairwise subsumption relations between these classes (which are also reduced to satisfiability/consistency tests) in order to build the class hierarchy of an ontology. In practice, the number of required satisfiability tests can be significantly reduced by optimised classification approaches such as enhanced traversal (Baader et al., 1994) or known/possible set classification (Glimm et al., 2012). These optimised classification algorithms use specific testing orders and exploit information that can be extracted from the constructed models. To optimise their testing order, the algorithms are usually initialised with told subsumptions, i.e., with the subsumption relations that can syntactically be extracted from ontology axioms, and, typically, the more told subsumers can be extracted, the larger is the benefit for the classification algorithms. However, a more detailed extraction of told subsumers from ontology axioms is usually less efficient than a simple one. For instance, the ontology axioms $A_1 \sqsubseteq \exists r.C \sqcap D$ and $\exists r.C \sqsubseteq A_2$ imply that A_2 is a subsumer of A_1, but this can only be detected, when parts of axioms are compared with each other.

With the saturation, we can significantly improve the told subsumers for the initialisation of the tableau-based classification algorithm since also (some) semantic consequences are considered. As new and more accurate told subsumers, we can simply use, for each concept A that has to be classified, all the (atomic) concepts in $\mathcal{L}(v_A)$. Moreover, if v_A is clashed, then we know that A is unsatisfiable without performing a satisfiability test. Analogously, if v_A is neither clashed nor critical and the knowledge base is consistent, we know that A is satisfiable and that $\mathcal{L}(v_A)$ contains all subsumers. Note that if v_A is nominal dependent and a representative node for a nominal is critical, then also v_A is identified as

critical. Hence, for the extraction of subsumers, we only have to consider the criticality status of the considered node, whereas nominal dependency does not matter. If no node for an ontology is critical, we already get all subsumers from the saturation and, therefore, only a transitive reduction (i.e., the elimination of those subsumptions that are indirectly implied through the transitivity property of the subsumption relation) is necessary to build the class hierarchy. Thus, with the preceding saturation we automatically get a one-pass classification for simple ontologies.

Note that our completeness proof in Section 3.3 principally only covers the satisfiability of concepts. However, it is quite obvious that the presented saturation approach also computes all subsumers if the nodes are neither critical nor clashed. In particular, if we assume that a subsumption $A \sqsubseteq B$ is not derived by the saturation but follows from a knowledge base $\mathcal{K}$, then we obtain a contradiction by considering the saturation of the knowledge base $\mathcal{K}'$ that extends $\mathcal{K}$ by the axiom $B \sqsubseteq \bot$. Since $B \sqsubseteq \bot$ can only be added with the $\sqsubseteq_1$-rule if B is in a node label and $\mathcal{K}'$ differs from $\mathcal{K}$ only through the axiom $B \sqsubseteq \bot$, the saturation would derive the same consequences and, hence, would be incomplete w.r.t. testing the satisfiability of A, which is contradictory w.r.t. our assumption and our completeness proof.

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 $D \sqsubseteq C$. As described in Section 2.2, we utilise this to get a normalised knowledge base where we do not have to consider such axioms. 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 has to be internalised into $\top \sqsubseteq \mathsf{nnf}(\neg D \sqcup A)$. 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 $\mathsf{nnf}(\neg D)$ (exploiting that $D \sqsubseteq A$ is equivalent to $\neg A \sqsubseteq \mathsf{nnf}(\neg D)$) (Horrocks & Tobies, 2000).[2] Unfortunately, using such an unfolding rule also comes at a price since the tableau algorithm is no longer forced to add either A or $\mathsf{nnf}(\neg D)$ 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 completion graphs do not contain A (and also not $\neg A$), which is an important optimisation for classification procedures (Glimm et al., 2012).

To compensate for this, we can create a "candidate concept" A^+ for A, for example by partially absorbing D (Steigmiller et al., 2014b), which is then automatically added to a node label in the completion graph if the node is possibly an instance of A, i.e., the candidate concepts indicate which completely defined concepts are possibly satisfied. 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 such a node (even if we allow the knowledge base to contain concept equivalence axioms of the form $A \equiv D$). Formally, we can define the requirements on such candidate concepts as follows:

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

Definition 14 (Candidate Concept)**.** *Let $\mathcal{K}$ be a knowledge base containing a complete definition of the form $A \equiv D$. We say that A^+ is a* candidate concept for *A if for every fully expanded and clash-free completion graph $G = (V, E, \mathcal{L}, \dot{\neq})$ (fully saturated saturation graph $S = (V, E, \mathcal{L})$), it holds that $A^+ \in \mathcal{L}(v)$ if $\mathcal{K} \models C_1 \sqcap \ldots \sqcap C_n \sqsubseteq A$, where $\{C_1, \ldots, C_n\} = \mathcal{L}(v)$ ($\{C_1, \ldots, C_n\} = \mathcal{L}(v)$ and v as well as the nodes representing individuals are neither critical nor clashed).*

Of course, with axioms of the form $\top \sqsubseteq A^+$, we can enforce that A^+ is added to all node labels and, hence, it represents a valid candidate concept for A. To be useful in practice, it is, however, desired that such concepts are added to as few node labels as possible (without introducing additional overhead) and, therefore, reasoners usually employ sophisticated absorption techniques to generate "better" candidate concepts (Steigmiller et al., 2014b). As a consequence, they are very handy for the identification of non-subsumptions as illustrated in the following example.

Example 3. *Let us assume that the TBox $\mathcal{T}_2$ consists of the axioms*

$$A_1 \sqsubseteq \exists r.B \qquad A_2 \sqsubseteq \exists s.B \sqcap (\forall r.\bot \sqcup \neg B) \qquad A_3 \equiv \exists s.B \sqcap \forall r.B,$$

and we are interested in the classification of $\mathcal{T}_2$. In order to get an (automatic) indication of concepts that could be subsumed by the completely defined concept A_3, we create a candidate concept for A_3 by (partially) absorbing the negation of A_3's definition. Hence, by partially absorbing $\forall s.\neg B \sqcup \exists r.\neg B$ to $B \sqsubseteq \forall s^-.A_3^+$ (the part $\exists r.\neg B$ cannot be absorbed trivially), we obtain the candidate concept A_3^+ for A_3. Note that this absorption only adds $B \sqsubseteq \forall s^-.A_3^+$ to $\mathcal{T}_2$ without removing or rewriting other axioms. Now, the absence of A_3^+ in a label indicates that A_3 is not a subsumer of the concepts in this label, which can be used for classification. In particular, if we saturate A_1, A_2, A_3, and B, then $\forall s^-.A_3^+$ is added to the label of the representative node for B and A_3^+ is propagated to the representative nodes for A_2 and A_3. In particular, since the label of the representative node for A_1 does not contain A_3^+, we know that $A_1 \sqsubseteq A_3$ does not hold without any special consideration of the axiom $A_3 \equiv \exists s.B \sqcap \forall r.B$. However, we still have to determine whether A_2 is satisfiable and which concepts are the subsumers of A_2 to complete the classification of $\mathcal{T}_2$. For this, we (have to) fall back to the tableau algorithm and, in principle, we have to perform a satisfiability test for A_2 and a subsumption test for each possible subsumer of A_2, i.e., for the (atomic) concepts that are (possibly non-deterministically) added to the root node in the satisfiability test, we have to checks whether they are actual subsumers in all models.

Although the candidate concepts already allow for a significant pruning of subsumption tests, there are still ontologies where these candidate concepts are added to many node labels, especially if only a limited absorption of D for an axiom of the form $A \equiv D$ is possible. Hence, A can still be a possible subsumer for many concepts.

The saturation graph can, however, again be used to 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 merging v with the saturated node $v_{\mathsf{nnf}(\neg D)}$ is possible. Since D is often a conjunction, we can also try to merge v with the representative node for a disjunct of $\mathsf{nnf}(\neg D)$. If the "models" can be "merged" as defined below, then v is obviously not an instance of A.

Definition 15 (Model Merging)**.** *Let $S = (V, E, \mathcal{L})$ be a fully saturated saturation graph and $G = (V', E', \mathcal{L}', \dot{\neq})$ 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;*
- *$\mathcal{L}(v) \cup \mathcal{L}'(v')$ does not contain $\{C, \mathsf{nnf}(\neg C)\}$ for some concept C;*
- *$\mathcal{L}(v) \cup \mathcal{L}'(v')$ does not contain concepts A_1 and A_2 such that $A_1 \sqcap A_2 \sqsubseteq C \in \mathcal{K}$ and $C \notin (\mathcal{L}(v) \cup \mathcal{L}'(v'))$;*
- *v' is not an r-neighbour of v' for a concept $\neg\exists r.\mathsf{Self} \in \mathcal{L}(v)$;*
- *v is not an r-neighbour of v for a concept $\neg\exists r.\mathsf{Self} \in \mathcal{L}'(v')$;*
- *$C \in \mathcal{L}'(w')$ for every r-neighbour w' of v' and $\forall r.C \in \mathcal{L}(v)$;*
- *$C \in \mathcal{L}(w)$ for every r-successor w of v and $\forall r.C \in \mathcal{L}'(v')$;*
- *$\mathsf{nnf}(\neg C) \in \mathcal{L}'(w')$ for every r-neighbour w' of v' and $\leqslant m\, r.C \in \mathcal{L}(v)$; and*
- *$\mathsf{nnf}(\neg C) \in \mathcal{L}(w)$ for every r-successor w of v and $\leqslant m\, r.C \in \mathcal{L}'(v')$.*

Note that the conditions are designed such that they can be checked very efficiently and it is clear that some of the conditions can be relaxed further. For instance, it is not necessary to enforce that v is not nominal dependent. In principle, we only have to ensure that there is no interaction with the generated completion graph, which can, for example, also be guaranteed if the concept tested for satisfiability does not use nominals in the completion graph. In addition, if the model merging fails due to concepts in the completion graph that have an interaction with the tested node in the saturation graph, then we can simply extend the saturation graph with a new node, where also the problematic concepts are considered, and retest the model merging with this node. For instance, if a node v' in the completion graph is not mergeable with a node v in the saturation graph due to an axiom $A_1 \sqcap A_2 \sqsubseteq C$ in the knowledge base for which $A_1 \in \mathcal{L}'(v')$, $A_2 \in \mathcal{L}(v)$, and $C \notin (\mathcal{L}'(v') \cup \mathcal{L}(v))$, then we can saturate a new node w with $\mathcal{L}(w) \supseteq \mathcal{L}(v) \cup \{C\}$ and check whether w is mergeable.

In contrast, if concepts from the tested node in the saturation graph interact with the completion graph, then it is often not easily possible to extend the model merging approach such that non-subsumption can be guaranteed. In particular, we are not interested in modifying the completion graph since it also has to be used for other model merging tests. In addition, a recursive model merging test, where we check whether the neighbours of a node in the completion graph are mergeable with propagated concepts from the saturation graph, is non-trivial since we have to exclude interactions with already tested nodes. For example, if a node v' in the completion graph is not mergeable with a node v in the saturation graph due to an r-neighbour w' of v' and a concept $\forall r.C$ in the label of v for which $C \notin \mathcal{L}(w')$, then a recursive model merging could test whether w' is mergeable with v_C. However, it would also be necessary to guarantee that the merging of w' with v_C does not cause new consequences that are propagated back to v', which is especially non-trivial if there are several universal restrictions in the label of v that would affect w'.

Example 4. *To continue the classification of the TBox $\mathcal{T}_2$ from Example 3, we (have to) build a completion graph for A_2 with the tableau algorithm, which is straightforward. In particular, we can directly see that A_2 is satisfiable and only A_3 can be a possible subsumer of A_2 (since the candidate concept A_3^+ is propagated to the root node for A_2 from the existentially restricted s-successor with B in its label). To apply model merging, the saturation of the different alternatives/disjuncts that correspond to $\neg A_3$ is required, i.e., we now assume that the concepts $\forall s.\neg B$ and $\exists r.\neg B$ have also been saturated, which is trivial since no new consequences are implied and all created and referred nodes can completely be handled by the saturation. If the tableau algorithm has added the disjunct $\forall r.\bot$ to satisfy $\forall r.\bot \sqcup \neg B$, then the model merging fails since $v_{\forall s.\neg B}$ has an interaction with the r-successor in the completion graph that has been constructed to satisfy $\exists s.B$ and for $v_{\exists r.\neg B}$ an interaction with $\forall r.\bot$ can obviously not be excluded. Hence, it would be required to test whether A_3 is a subsumer of A_2 by checking the satisfiability of $A_2 \sqcap \neg A_3$. In contrast, if $\neg B$ has been added, then none of the model merging conditions are satisfied for $v_{\exists r.\neg B}$ and, therefore, we can directly conclude that A_3 is not a subsumer of A_2.*

Note, although other proposed (pseudo) model merging techniques (Haarslev, Möller, & Turhan, 2001) work, in principle, in a very similar way, there are also some significant differences. For example, the presented merging test is only applied if corresponding candidate concepts are in the label of nodes, which already reduces the number of tests. In addition, we test the merging against nodes from the saturation graph and, therefore, we do not have any significant overhead in creating appropriate (pseudo) models. In contrast, for other approaches it is often necessary to build separate completion graphs for those concepts for which the model merging is to be applied. Moreover, the presented approach is also applicable to very expressive DLs such as $\mathcal{SROIQ}$, whereas other approaches often deactivate model merging if certain language features are used (e.g., nominals). Of course, very expressive DLs may produce more critical nodes and, therefore, they potentially reduce the model merging possibilities, but it is not necessary to completely deactivate it, which results in a very good pay-as-you-go behaviour.

5. Saturation Improvements

Obviously, the support of the tableau algorithm with the saturation works better when as few nodes as possible are marked critical. However, since our saturation procedure does not completely support all language features, we easily get critical nodes even when the unsupported language features are only rarely used in the knowledge base. This is especially problematic if the critical nodes are referenced by many other nodes, whereby they also have to be considered critical. In the following, we present different approaches about how the saturation can be improved such that the number of critical nodes can be reduced. As a result, a better support of the tableau algorithm is possible.

5.1 Supporting More Expressive Language Features

As known from the literature, saturation procedures can be extended to more expressive Horn DLs, e.g., Horn-$\mathcal{SHIQ}$ (Kazakov, 2009) or even Horn-$\mathcal{SROIQ}$ (Ortiz, Rudolph, & Simkus, 2010). Although it has been shown that such extensions can be very efficient for

ontologies in these fragments, it is not completely clear how they perform for ontologies that use language features outside of these fragments, for example, if they are used to partially saturate ontologies as for our approach. In particular, the worst-case complexity for such procedures is not polynomial and, therefore, they can easily cause the construction of very large saturation graphs with corresponding large memory requirements. However, in practical implementations, we can simply limit the number of nodes that are processed by the saturation by directly marking the remaining nodes as critical. Hence, we can easily support some features of such Horn-languages without risking that the memory consumption is increased too much without gaining some benefits.

In particular, it is very interesting to relax the restriction that concepts are only propagated to predecessor nodes for universal restrictions of the form $\forall r.C$. This is required for the saturation procedure presented in Section 3 to enable the reuse of nodes, but it can be extended such that a full support of universal restrictions is possible. Of course, we are not allowed to directly modify existing r-successors, but we can easily create and saturate copies of the existing r-successors that we extend by the propagated concept C. In addition, we can remove the edges to the previous r-successors such that the incompleteness detection rule $\mathcal{C}_{\downarrow\forall}$ for the concept $\forall r.C$ does not mark the node as critical, which is obviously not the case if now all newly connected r-successors include the concept C and are completely handled. Note that these copies and extensions of nodes can be realised very efficiently. Basically, we first apply the default saturation rules and, afterwards, we extend only those successors where the saturation has not already added the concept C. In addition, we can use, for each successor node that has to be extended, a mapping for the concepts, for which the node has to be extended, to the copied and extended nodes, whereby we can reuse already created node extensions. Thus, if several predecessors propagate the same concepts to the same successors, then we create a node with the corresponding extension only once. This can be seen as an (efficient) implementation of the so-called *node contexts*, which serve as basis for many saturation procedures that can fully handle universal restrictions (Simančík et al., 2011, 2014; Bate et al., 2015). In particular, our "extension mapping", i.e., the mapping from nodes to copies of the nodes extended by the additional concepts, can be seen as a representation of such node contexts. For example, if we have a node v_A as an r-successor of v and $\forall r.B \in \mathcal{L}(v)$, then we create a copy of the node v_A, say $v_{A,B}$, for which B is added and which is then used as r-successor of v instead of v_A. With our extension mapping, we then also store that the extension of v_A by B can be found in the node $v_{A,B}$ such that we can reuse it. Note, however, that we create the copy only if B is not already in the label of v_A. Moreover, by directly copying the nodes (with the derived consequences), a repetition of many rule applications is not necessary.

Support for at-most restrictions of the form $\leqslant 1 r.\top$ can be achieved analogously. The labels of corresponding r-successors can easily be merged into a new node, which can then be used to replace the other r-successors. Again, we can use a mapping such that the merging of certain successors always results in the same (possibly new) node. If there is a remaining r-successor v' that also has to be merged to a predecessor v'' for a node v, then we add all the concepts in the label of v' to the label of v'' and we make v also an $\mathsf{inv}(r)$-successor of v''. Thus, Horn-$\mathcal{SHIF}$ can (almost) completely be supported with rather small extensions of the presented saturation procedure.

More difficult is the support of nominals. Already a complete nominal support for the DL $\mathcal{EL}^{++}$ would potentially introduce some significant overhead. In particular, it would be necessary to store, for every node v and every nominal $\{a\}$, which descendants of v are using the nominal $\{a\}$, i.e., if a descendant of v has a nominal $\{a\}$ in its label, then we would have to store for v that the nominal $\{a\}$ is used by this descendant. If we would find a node v, for which it is stored that a nominal $\{a\}$ is used by several descendant nodes, say $v^1, \ldots, v^n$, then we would have to create a new node u where the labels of $v^1, \ldots, v^n$ were merged, and we would have to "reproduce" the paths of predecessors from the merged nodes up to v such that potentially new consequences can also be propagated to v. However, since the majority of EL ontologies use nominals only in much simpler ways (e.g., with concepts of the form $\exists r.\{a\}$) for which the presented saturation procedure is already sufficient, a more sophisticated nominal handling does currently not seem to be required.

Saturation procedures can further be extended to non-Horn DLs, for instance, saturation procedures have been proposed for the DLs $\mathcal{ALCH}$ (Simančík et al., 2011), $\mathcal{ALCI}$ (Simančík et al., 2014), and even for $\mathcal{SHIQ}$ (Bate et al., 2015). For this, they also have to handle non-determinism that is, for example, caused by disjunctions, which is typically realised by simply considering/saturating all non-deterministic alternatives. If the same concepts are derived for all alternatives, then they are interpreted as actual consequences of the knowledge base. If the number of alternatives is very large, then such a (naive) saturation approach might become impractical. Although also the tableau algorithm has to consider all alternatives in the worst-case, it is doing it successively, i.e., it is trading memory requirements against a potentially increased runtime. Moreover, tableau algorithms usually implement a large amount of optimisations to reduce the non-deterministic alternatives that have to be considered. Most notably, dependency directed backtracking allows for evaluating only those alternatives of non-deterministic decisions that are indeed relevant, i.e., which are involved in the creation of clashes. Since saturation algorithms do not track dependencies between derived facts, their ability to determine which alternatives of non-deterministic decisions do not have to be considered (since they would result in the same clashes) is very limited. Unfortunately, the tracking of dependency information makes a simple reuse of nodes (which is the foundation of saturation procedures) impossible or, at least, much more involved.

Although it has been shown that saturation procedures extended to non-deterministic language features can work very well for a range of ontologies (Simančík et al., 2011), more investigations are required in order to understand whether (or in which cases) they are better than tableau algorithms for more expressive DLs. However, the development and the implementation of saturation-based reasoning systems for more expressive DLs seems challenging and, to the best of our knowledge, a saturation-based procedure/reasoner for expressive DLs such as $\mathcal{SROIQ}$ does not yet exist. Hence, it can be an interesting compromise, as presented in this paper, to keep the (basic) saturation algorithm deterministic and to process the remaining parts with the tableau algorithm, which is typically coupled with several well-established optimisations (e.g., semantic branching, Boolean constraint propagation, dependency directed backtracking, unsatisfiability caching) to handle non-determinism. Alternatively, one can process non-deterministic language features with the saturation procedure only as long as certain limits are not reached (e.g., a memory limit or

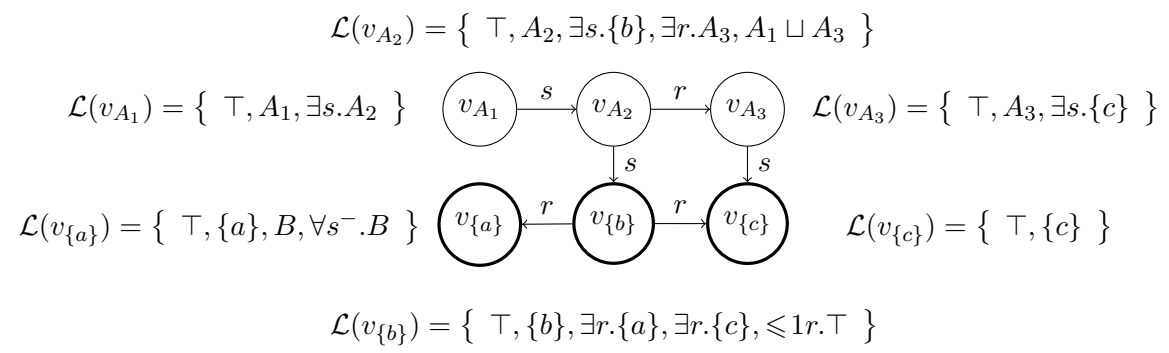


Figure 2: Incompletely handled saturation graph for testing the satisfiability of the concept A_1 from Example 5

an upper bound for the number of saturated, non-deterministic alternatives), and can simply mark remaining nodes as critical such that they are processed by the tableau algorithm.

5.2 Improving Saturation with Results from Completion Graphs

As already mentioned, even if there is only one node for an individual that is critical, then the presented saturation procedure also marks all nominal dependent nodes as critical. This easily limits the improvement from the saturation for ontologies that intensively use nominals. Analogously, if there are few nodes with incompletely handled concepts (e.g., disjunctions) and these nodes are referenced by many other nodes, then all these other nodes are also critical although they do not necessarily have concepts in their label that cannot be handled completely. Both issues are also illustrated in the following example:

Example 5. *Let us assume that the TBox $\mathcal{T}_3$ contains the following axioms:*

$$\begin{array}{llll} A_1 \sqsubseteq \exists s.A_2 & A_2 \sqsubseteq \exists s.\{b\} & A_2 \sqsubseteq \exists r.A_3 & A_2 \sqsubseteq A_1 \sqcup A_3 \\ A_3 \sqsubseteq \exists s.\{c\} & B \sqsubseteq \forall s^-.B & & \\ \{a\} \sqsubseteq B & \{b\} \sqsubseteq \exists r.\{a\} & \{b\} \sqsubseteq \exists r.\{c\} & \{b\} \sqsubseteq \leqslant 1r.\top \end{array}$$

For testing the satisfiability of the concept A_1 w.r.t. TBox $\mathcal{T}_3$, we generate the saturation graph that is depicted in Figure 2. Note, the node $v_{\{b\}}$ for the individual b cannot be completely handled by the saturation due to the concept $\leqslant 1r.\top$ in the label of $v_{\{b\}}$, which would require that $v_{\{a\}}$ and $v_{\{c\}}$ are merged. Therefore, $v_{\{b\}}$ is critical and we also have to consider all nodes as critical that refer to such critical nodes, which is, for example, the case for the node v_{A_2}. Moreover, since one node for an individual is critical, we cannot exclude that more consequences are propagated to other individuals and, therefore, possibly also to other nominal dependent nodes. For instance, the merging of $v_{\{a\}}$ and $v_{\{c\}}$ would propagate the concept B to the label of v_{A_3}. Thus, also v_{A_3} is critical although it does not directly contain a concept that cannot be handled by the saturation. Analogously, the label of v_{A_2} contains the disjunction $A_1 \sqcup A_3$, which is also not completely processed by the

saturation and, therefore, we have to mark all ancestor nodes of v_{A_2} as critical (if this is not already the case), even if they do not contain problematic concepts. As a consequence, we obtain a saturation status $\mathcal{S} = (\mathcal{S}_\mathsf{o}, \mathcal{S}_{\not<}, \mathcal{S}_!)$, where $v_{\{b\}}$ has a tight at-most restriction, i.e., $\mathcal{S}_{\not<} = \{v_{\{b\}}\}$, and all nodes are nominal dependent as well as critical, i.e., $\mathcal{S}_\mathsf{o} = \mathcal{S}_! = \{v_{\{a\}}, v_{\{b\}}, v_{\{c\}}, v_{A_1}, v_{A_2}, v_{A_3}\}$.

Of course, the saturation can be extended in several ways to better support features of more expressive DLs (see Section 5.1), but, to the best of our knowledge, there exists no saturation algorithm that completely covers all the features of very expressive DLs such as $\mathcal{SROIQ}$. Hence, if a knowledge base uses some of the unsupported features, then we easily run into the problem that the saturation becomes incomplete and we possibly get many critical nodes.

An approach to overcome the issues with critical nodes is to "patch", i.e., update, the saturation graph with results from fully expanded and clash-free completion graphs that are generated for consistency or satisfiability checks. Roughly speaking, the idea is to replace the labels of critical nodes in the saturation graph with corresponding labels from these completion graphs, for which we know that they are completely handled by the tableau algorithm. We call such nodes *patched* nodes. Then, we apply the saturation rules again and we update the saturation status, which hopefully results in an improved saturation graph with fewer critical nodes. Note, however, that simply adding non-deterministically derived concepts from labels of completion graphs to the saturation easily leads to unsound results. Hence, we distinguish deterministically and non-deterministically derived concepts when updating the saturation by simultaneously managing two saturation graphs: one where only the deterministically derived concepts are added, i.e., the "deterministic saturation graph", and a second one, where also the non-deterministically derived concepts and consequences are considered, i.e., the "non-deterministic saturation graph". 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, which can then be considered as non-critical. By reducing the number of critical nodes in the saturation, this approach then allows for further improving the construction of new completion graphs by transferring new and more results from the updated saturation.

In order to describe the approach in more detail, we first define a saturation patch, which constitutes the data structure for managing the information that is necessary for updating a saturation graph.

Definition 16 (Saturation Patch)**.** *Let $\mathsf{fclos}(\mathcal{K})$ ($\mathsf{Rols}(\mathcal{K})$) denote the concepts (roles) that possibly occur in completion graphs for the knowledge base $\mathcal{K}$ as defined in Definition 5. A* saturation patch *P for a saturation graph $S = (V, E, \mathcal{L})$ w.r.t. $\mathcal{K}$ is a tuple $P = (V_p, \mathcal{L}_d, \mathcal{L}_n, M_c, V_o)$, where*

- *$V_p \subseteq V$ denotes the set of patched nodes in the saturation graph,*
- *$\mathcal{L}_d \colon V_p \to 2^{\mathsf{fclos}(\mathcal{K})}$ is the mapping of patched nodes to a set of deterministically derived concepts,*

- $\mathcal{L}_n \colon V_p \to 2^{\mathsf{fclos}(\mathcal{K})}$ *is analogously the mapping of patched nodes to a set of non-deterministically derived concepts,*
- $M_c \colon V_p \times \mathsf{Rols}(\mathcal{K}) \times \mathsf{fclos}(\mathcal{K}) \to \mathbb{N}_0$ *is the mapping of at-most cardinality restrictions of the form* $\leqslant m\, r.C$ *on patched nodes (represented as a tuple of the node* v*, the role* r*, and the qualification concept* C*) to the number of merging candidates, and*
- $V_o \subseteq V_p$ *denotes the patched nodes that are nominal dependent.*

A saturation patch obviously has to identify the nodes that should be patched/updated, which is realised with the set V_p. For each node in V_p, the mappings $\mathcal{L}_d$ and $\mathcal{L}_n$ contain the concepts from the node's label in the completion graph that are derived deterministically and non-deterministically, respectively. Hence, these mappings determine how a node's label in the saturation graph can be extended such that it is no longer critical. We also have to store the number of merging candidates (M_c) and the patched nodes that are nominal dependent (V_o) because this information is required for the generation of an updated saturation status. Note that we consider the number of merging candidates and the nominal dependencies as non-deterministic information since it is often not possible to correctly extract the corresponding deterministic information from completion graphs. For example, state-of-the-art reasoners are usually searching blocker nodes by checking more detailed conditions as defined for pairwise blocking, whereby a node can possibly also be blocked if the label is a subset of the label from the blocker node (Horrocks & Sattler, 2001). If the blocker node is directly or indirectly using nominals, i.e., it is nominal dependent, then also the blocked node has to be considered as nominal dependent. Hence, we have to consider the nominal dependency as non-deterministic information since the nominal dependency could be caused by a concept that is in the label of the blocker node but not in the label of the blocked one.

Especially the root nodes of completion graphs constructed for satisfiability and consistency tests are very suitable for the extraction of patches. For instance, in a fully expanded and clash-free completion graph for testing the satisfiability of a concept C, the root node has C in its label and can be used as a patch for the node v_C in the saturation graph. The completion graph of a consistency check can be used to patch representative nodes for nominals. Of course, for the patching of nominal dependent nodes, we have to ensure some kind of consistency, i.e., the dependent nominals have to be compatible with the representative nodes of these nominals in the saturation graph and the already applied patches for these nodes. A simple form of compatibility can be defined as follows:

Definition 17 (Saturation Patch Compatibility)**.** *Let* $P^1 = (V_p^1, \mathcal{L}_d^1, \mathcal{L}_n^1, M_c^1, V_o^1), \ldots, P^n = (V_p^n, \mathcal{L}_d^n, \mathcal{L}_n^n, M_c^n, V_o^n)$ *be saturation patches for a saturation graph* S *w.r.t. a knowledge base* $\mathcal{K}$*, where* $V_p^i = \{v_1^i, \ldots, v_{m^i}^i\}$ *for* $1 \leq i \leq n$*. We say that* $P^1, \ldots, P^n$ *are* compatible *if a fully expanded and clash-free completion graph* $G = (V, E, \mathcal{L}, \dot{\neq})$ *can be built for* $\mathcal{K}$ *such that it contains the nodes* $w_1^1, \ldots w_{m^1}^1, \ldots, w_1^n, \ldots w_{m^n}^n$ *with* $\mathcal{L}(w_j^i) = \mathcal{L}_d^i(v_j^i) \cup \mathcal{L}_n^i(v_j^i)$ *for* $1 \leq j \leq m^i$ *and* $1 \leq i \leq n$*.*

In principle, we are not limited to the root and nominal nodes for the extraction of patches, but a more detailed analysis of the completion graph is required for other nodes. For example, the tableau algorithm does not apply the $\exists$-rule for a concept $\exists r.C$ in the

label of a node v if v already has an r^--predecessor v' with C in its label. Hence, if the predecessor v' directly or indirectly uses nominals, then also v has to be considered as nominal dependent. Moreover, for other nodes in the completion graph, it is often not clear which concepts have to be considered as non-deterministically derived consequences. For instance, if we create, for the concepts $\exists r.C$ and $\forall r.D$ in the label of a node v, the r-successor v' and we extract a patch for v_C from v', then D has to be identified as a non-deterministically derived concept. For this, it is in principle necessary to track and analyse the dependencies between facts and their causes in the completion graph. If this is efficiently supported by a reasoning system, then the extraction of patches can also be extended to other nodes in the completion graph. Otherwise, the patch creation can simply be restricted as appropriate.

The saturation patches are applied to a saturation graph as follows:

Definition 18 (Saturation Patch Application)**.** *Let $S = (V, E, \mathcal{L})$ be a saturation graph and $P = (V_p, \mathcal{L}_d, \mathcal{L}_n, M_c, V_o)$ a saturation patch for S. The* deterministic (non-deterministic) application *of P to S yields a* deterministically (non-deterministically) extended saturation graph *S_d (S_n) of S that is obtained by saturating the saturation graph $(V, E, \mathcal{L}')$, where $\mathcal{L}' = \{v \mapsto \mathcal{L}(v) \mid v \in V \setminus V_p\} \cup \{v \mapsto \mathcal{L}_d(v) \mid v \in V_p\}$ ($\mathcal{L}' = \{v \mapsto \mathcal{L}(v) \mid v \in V \setminus V_p\} \cup \{v \mapsto \mathcal{L}_d(v) \cup \mathcal{L}_n(v) \mid v \in V_p\}$).*

Since we are interested in a deterministic and in a non-deterministic saturation graph, we create a copy of the saturation graph as soon as we have a patch with non-deterministically derived concepts and, then, we use the non-deterministic application of patches only for the copy. Although we can also fully saturate the non-deterministic saturation graph by simply using the presented saturate function, this potentially derives unwanted consequences since the application of all saturation rules for all nodes possibly propagates new consequences to patched nodes. This can be unfavourable if patched consequences are derived from the processing of different non-deterministic alternatives. In particular, if a node v and an ancestor of v are patched, then the saturation rules might propagate new consequences obtained from the patching of v up to the ancestors. If an ancestor is, however, patched with concepts from another completion graph, where different non-deterministic alternatives are processed, then we possibly mix consequences of different alternatives in the saturation graph, which easily limits the effectiveness of our approach. For example, if v contains the disjunction $\forall r.A \sqcup \forall r.\neg A$, and we patch v with the non-deterministic extension $\forall r.A$, then the patching of the r^--predecessor v' of v with the non-deterministic extension $\neg A$ allows the application of the $\forall$-rule for the concept $\forall r.A$ in the label of the node v such that the concept A is propagated to v'. As a consequence, we would infer with the saturation that v' is (possibly) clashed since A and $\neg A$ are in its label. Since all (new) consequences in the non-deterministic saturation graph are considered to be non-deterministic, this does not produce incorrect results. In order to, nevertheless, avoid the derivation of such unwanted consequences, we can saturate the non-deterministic saturation graph that contains the patched nodes V with a modified $\mathsf{saturate}^{\setminus V}(S)$ function, where only the $\forall$-rule is applied to the nodes in V and the $\forall$-rule is modified such that it does not propagate concepts to a node $v \in V$. For this (and for a more precise detection of the saturation status), we gather all patches in one combined patch and keep this patch in addition to the deterministic

and non-deterministic saturation graph. The patches can straightforwardly be combined by using the $\circ$-operator defined as follows:

Definition 19 (Saturation Patch Composition)**.** *Given two saturation patches P and P' with $P = (V_p, \mathcal{L}_d, \mathcal{L}_n, M_c, V_o)$ and $P' = (V_p', \mathcal{L}_d', \mathcal{L}_n', M_c', V_o')$, the saturation patch $P \circ P'$ is defined as the tuple consisting of*

- $V_p \cup V_p'$,
- $\mathcal{L}_d \cup \{v \mapsto \mathcal{C} \mid v \mapsto \mathcal{C} \in \mathcal{L}_d' \text{ and } v \notin V_p\}$,
- $\mathcal{L}_n \cup \{v \mapsto \mathcal{C} \mid v \mapsto \mathcal{C} \in \mathcal{L}_n' \text{ and } v \notin V_p\}$,
- $M_c \cup \{\langle v, s, C\rangle \mapsto n \mid \langle v, s, C\rangle \mapsto n \in M_c' \text{ and } v \notin V_p\}$, *and*
- $V_o \cup (V_o' \setminus V_p)$.

Note that if both patches contain information about the same node, then we keep the information for this node only from one (the new) patch instead of mixing the information. Thus, the information from the other patch gets lost for all common nodes, which is, however, not problematic since both patches describe valid extensions.

In addition to the modified saturation function, we should reprocess the ancestors for patched nodes in the non-deterministic saturation graph if the patching removes previously added non-deterministic consequences in order to avoid the mixing of consequences from different non-deterministic alternatives. For example, if a non-deterministically derived concept such as $\forall r.C$ was added or propagated to a node label and is removed by patching this node, then the r^--predecessor should also be rebuilt from the deterministic saturation graph such that unnecessary non-deterministic consequences (e.g., C) can also be removed. For practical implementations, we can obviously limit the number of ancestor nodes that are updated or processed for new non-deterministic consequences in the non-deterministic saturation graph in order to limit the overhead of the patch application. If the limit is reached, then the remaining ancestors can simply be marked as critical. Also note that we can reuse all the data of the deterministic saturation graph in the non-deterministic one for nodes that are not influenced by a patch with non-deterministic consequences.

In order to be able to use patched saturation graphs for the support of the tableau algorithm, e.g., for the transfer of results into completion graphs, we have to update the saturation statuses after the application of the patches. Similarly to the rule application of the non-deterministic saturation graph, we do not want to propagate a status to a patched node from the successors. Therefore, we analogously use a modified $\mathsf{status}^{\setminus V}$ function instead of status, where the rules of Table 3 and 4 are only applied to nodes that are not in V. This also requires that we use a modified $\sharp\mathsf{mcands}'$ function in $\mathsf{status}^{\setminus V}$ since, for the patched nodes, we have to use the correct information as given by the patch. To be more precise, when M_c denotes the mapping to the number of merging candidates in the considered patch, then $\sharp\mathsf{mcands}'(v, s, D)$ has to return $M_c(\langle v, s, D\rangle)$ if v is a patched node, and $\sharp\mathsf{mcands}(v, s, D)$ otherwise. In addition, we have to correctly initialise the sets $\mathcal{S}_{\not<}$, $\mathcal{S}_{\mathsf{o}}$, and $\mathcal{S}_{!}$ for the patched nodes with the information from the applied patches. For the non-deterministic saturation graph, all patched nodes are obviously non-critical since their labels have been extracted from fully expanded and clash-free completion graphs. Hence,

we only have to initialise $\mathcal{S}_{\not\leqslant}$ and $\mathcal{S}_{\mathsf{o}}$ for a (combined) patch $P = (V_p, \mathcal{L}_d, \mathcal{L}_n, M_c, V_o)$, which can be realised by setting $\mathcal{S}_{\mathsf{o}} = \{v \mid v \in V_o\}$, and

$$\mathcal{S}_{\not\leqslant} = \{v \mid v \in V_p \text{ and } \leqslant m\, r.C \in (\mathcal{L}_d(v) \cup \mathcal{L}_n(v)) \text{ such that } M_c(\langle v, r, C\rangle) = m\}.$$

For the deterministic saturation graph, we additionally have to set $\mathcal{S}_!$ to $\{v \mid \mathcal{L}_n(v) \neq \emptyset\}$ in order to mark all patched nodes directly as critical if they could depend on non-deterministic consequences. After the initialisation, we can call the function $\mathsf{status}^{\setminus V_p}$ to obtain a full status for the corresponding saturation graph, which can then be used to further improve the support of the tableau algorithm.

Analogously to the deterministic and non-deterministic saturation graphs, every new saturation status can incrementally be updated from the last generated status for the last saturation graphs by sequentially updating the ancestors for the newly patched nodes. Hence, also the generation of new saturation statuses is not causing a significant overhead in practice.

The patching of saturation graphs enables a more sophisticated support of tableau algorithms. On the one hand, the patching reduces the number of critical nodes and, therefore, the optimisations described in Section 4, such as blocking the expansion of successors nodes in the completion graph or the extraction of subsumers, are better applicable. On the other hand, we can now also use the non-deterministic saturation graph for the support, e.g., in the classification process. If a node v_A in the non-deterministic saturation graph is not critical, then the label of v_A in the non-deterministic saturation graph describes all possible subsumers of A. Thus, if v_A is not critical in the non-deterministic saturation graph, then its label can be used to prune possible subsumers. Moreover, we can use the non-deterministic saturation graph to find identical labels that can be used for blocking the processing/expansion of successors nodes in the completion graph. Of course, we still require that the corresponding nodes in the (non-deterministic) saturation graph are not critical. In contrast, the restriction that the nodes in the saturation graph are not allowed to be nominal dependent for the blocking can be relaxed such that it works sufficiently well for many real world ontologies. Basically, we patch all nodes that represent individuals in the saturation graph after the consistency check with the corresponding nodes in the obtained fully expanded and clash-free completion graph. Furthermore, we ensure, on the one hand, that each subsequent saturation patch is compatible to this initial patch, i.e., we only create patches for nominal dependent nodes if all the labels of the nodes for the individuals in a completion graph are identical or subsets of the corresponding labels of the initial completion graph from the consistency check. On the other hand, we do not create patches for nodes if they depend on new nominals, i.e., on nominals that are introduced by the NN-rule. This ensures that the nodes in the saturation graphs can be used for blocking as long as we expand the nodes for the individuals in the same way as in the initial completion graph. Thus, if nominal dependent nodes are used for blocking, we collect the blocked nodes in a queue and we reactive these nodes if it becomes necessary to expand the nodes for the individuals in another way as in the completion graph for the initial consistency check. Of course, with a more exact tracking of the dependent nominals, e.g., by exactly saving on which nominals a node possibly depends, we can refine and improve this technique significantly. Obviously, if we use a node for blocking for which it is exactly known on which nominals it depends, then we only have to reactivate the processing of this node if the

nodes for the corresponding individuals are expanded differently. Although this approach keeps the patching of the saturation graphs consistent, i.e., the compatibility of the patches is automatically ensured, it is more restrictive than required in Definition 17. However, it allows for identifying potential incompatibilities with other techniques of tableau-based reasoning systems, e.g., with variants of completion graph caching techniques (Steigmiller, Glimm, & Liebig, 2015).

Due to the non-deterministic decisions of the tableau algorithm, a critical node in the saturation graph can be patched in several ways. Moreover, we can patch an already patched node to (hopefully) improve the non-deterministic saturation graph, i.e., we try to reduce the number of nodes that are influenced by the non-deterministic consequences and/or marked as critical. Thus, we need a strategy that decides for which nodes we have to extract patches from a fully expanded and clash-free completion graph such that the non-deterministic saturation graph can be improved. As already described, we can only extract patches from nodes for which all information can be safely extracted and they do not make the non-deterministic saturation graph inconsistent. In addition, the strategy has to keep the number of patches as small as possible since we have to update the data structures for every patch.

A simple example for such a strategy is to create only patches when they reduce the number of non-deterministic propagation concepts for the patched nodes. With this strategy we would prefer a patch that adds the non-deterministic set of concepts $\{\forall r.C, A_1, A_2\}$ in comparison with a patch that has the non-deterministic extension $\{\forall s.D, \forall t.D\}$. This strategy ensures, at least, that we do not create arbitrary patches, which avoids an oscillation between different possibilities, and we clearly favour the creation of patches that do not influence other nodes. However, we cannot guarantee that the non-deterministic saturation graph is actually improved. For example, the concept $\forall r.C$ could propagate C to several predecessors and also the processing of C could further influence many ancestors, whereas the patch with $\{\forall s.D, \forall t.D\}$ might only influence few predecessors. Therefore, if the node is already patched with $\{\forall s.D, \forall t.D\}$ and we create a new patch with $\{\forall r.C, A_1, A_2\}$ due to the fewer propagation concepts, then we even worsen the non-deterministic saturation graph. In order to counteract this, we should also extract patches from the saturation graph if we detect that a critical node in the deterministic saturation graph is labelled with the same concepts as in the non-deterministic saturation graph and the node in the non-deterministic saturation graph is not critical. With this kind of "internal patch" we can ensure that if the saturation has identified a node that is neither critical nor influenced by non-deterministic consequences, then we remember this "solved" state of the node and we do not overwrite its state by integrating other patches in the non-deterministic saturation graph. Of course, the strategy for the creation and extraction of patches optimally also considers the nominal dependency and tight at-most restrictions by trying to reduce the number of such nodes.

Example 6. *As mentioned, all nodes in the saturation graph of Figure 2, which is generated for testing the satisfiability of the concept A_1 w.r.t. TBox $\mathcal{T}_3$ (p. 564), are critical. As a consequence, we have to check the satisfiability of A_1 with the tableau algorithm in detail. For this, we first check the consistency of the individuals a, b, and c, which results in a simple completion graph, where the nodes for a and c are merged. From this completion graph, we extract an initial saturation patch P^1 for the individuals, i.e., $P^1 = (V_p^1, \mathcal{L}_d{}^1, \mathcal{L}_n{}^1, M_c^1, V_o^1)$*

with $V_p^1 = \{v_{\{a\}}, v_{\{b\}}, v_{\{c\}}\}$, $\mathcal{L}_d{}^1 = \{v_{\{a\}} \mapsto \{\top, \{a\}, \{c\}, B, \forall s^-.B\}, v_{\{c\}} \mapsto \{\top, \{a\}, \{c\}, B, \forall s^-.B\}, v_{\{b\}} \mapsto \{\top, \{b\}, \exists r.\{a\}, \exists r.\{c\}, \leqslant 1\, r.\top\}\}$, $\mathcal{L}_n{}^1 = \emptyset$, $M_c^1 = \{\langle v_{\{b\}}, r, \top\rangle \mapsto 1\}$, *and* $V_o^1 = \{v_{\{a\}}, v_{\{b\}}, v_{\{c\}}\}$. *Note, although the nodes for the individuals* a *and* c *are merged in the completion graph, we have to patch* $v_{\{a\}}$ *and* $v_{\{c\}}$ *separately since the saturation does not support the merging of nodes. Also note that the completion graph for the consistency check is deterministic and, therefore, the mapping of nodes to non-deterministically derived concepts is not required, i.e., each node has to be mapped to* $\emptyset$ *for* $\mathcal{L}_n{}^1$. *However, for ease of presentation, we omit uninteresting patch data and we simply use* $\emptyset$ *for* $\mathcal{L}_n{}^1$.

By deterministically applying P^1 *to our initial saturation graph, we obtain a new deterministic saturation graph, where the nodes are extended by the data from the applied patch. In particular,* $v_{\{c\}}$ *is extended by the concepts* $\{a\}, B$, *and* $\forall s^-.B$ *in this deterministic saturation graph, whereby the concept* B *is also propagated to* v_{A_3} *and, as a consequence, the label of* v_{A_3} *is extended to the set* $\{\top, A_3, \exists s.\{c\}, B, \forall s^-.B\}$. *The saturation status for the new deterministic saturation graph reveals that the nodes* $v_{\{a\}}$, $v_{\{b\}}$, $v_{\{c\}}$, *and* v_{A_3} *are not critical. Thus, we have, in principle, already shown the satisfiability of the concept* A_3. *In contrast,* v_{A_1} *is still indirectly critical due to the incompletely handled disjunction* $A_1 \sqcup A_3$ *in the label of* v_{A_2}.

In order to test the satisfiability of A_1 *now with the tableau algorithm, we initialise a new completion graph with a node* v *for which the concept* A_1 *is asserted. Since the disjunction* $A_1 \sqcup A_3$ *will be added to this completion graph for the* s*-successor* v' *of* v, *the tableau algorithm has to choose between the disjuncts* A_1 *and* A_3. *Independently from the decision, we can obtain a fully expanded and clash-free completion graph that shows the satisfiability of* A_1, *but the non-deterministic decision influences the patching of the saturation graph. For example, by non-deterministically adding* A_3 *to* v', *the tableau algorithm has to add an* s*-edge to the node representing* c *due to* $A_3 \sqsubseteq \exists s.\{c\} \in \mathcal{T}_3$ *and then* B *is propagated to the label of* v' *and, subsequently, also to the label of* v *due to* $B \sqsubseteq \forall s^-.B \in \mathcal{T}_3$ *and since the node for* c *has* B *in its label. Thus, we can extract a patch* $P^2 = (\{v_{A_1}\}, \{v_{A_1} \mapsto \{\top, A_1, \exists s.A_2\}\}, \{v_{A_1} \mapsto \{B, \forall s^-.B\}\}, \emptyset, \{v_{A_1}\})$. *Since* P^2 *contains non-deterministically derived consequences, we apply the patch deterministically and non-deterministically. Although the node* v_{A_1} *can be considered as fully handled in the non-deterministic saturation graph, it remains critical in the deterministic saturation graph and, therefore, its usage for supporting (e.g., blocking the expansion of successor nodes in new completion graphs, identification of (possible) subsumers) the tableau algorithm is limited. In contrast, if the disjunct* A_1 *were non-deterministically added to* v', *then we could extract a saturation patch* $P^3 = (\{v_{A_1}\}, \{v_{A_1} \mapsto \{\top, A_1, \exists s.A_2\}\}, \emptyset, \emptyset, \{v_{A_1}\})$ *and by applying* P^3, *we could also consider the node* v_{A_1} *as non-critical in the deterministic saturation graph. Hence, we prefer the saturation patch* P^3 *and we would also extract and apply* P^3, *even if we extracted and applied* P^2 *from an earlier constructed completion graph.*

As of now, we only considered patching from fully expanded and clash-free completion graphs. Of course, it is also possible to integrate unsatisfiability results from completion graphs into the saturation graphs. In particular, if the tableau algorithm cannot find a fully expanded and clash-free completion graph for a concept C, then we can create a patch where we deterministically extend v_C by the concept $\bot$. Such a management of unsatisfiable

concepts with the saturation graph has the benefit that $\bot$ is also propagated to other nodes and we can immediately identify many other unsatisfiable concepts.

It is also worth pointing out that, especially with the extraction and application of patches, the support of the tableau algorithm with the information provided through the saturation graphs can be seen as an intelligent caching technique. Although this only corresponds to a limited caching for certain nodes that are not further influenced by predecessors, it also works, to some extent, with nominals and inverse roles. Moreover, it is very fast and can automatically propagate unsatisfiability and satisfiability statuses to other concepts.

6. Related Work

There are already some approaches that combine the reasoning techniques of fully-fledged DL reasoners with specialised procedures for specific fragments. For instance, the reasoning system MORe (Armas Romero et al., 2012) uses module extraction to identify a part of an ontology that can be completely handled by a more efficient reasoning system and the fully-fledged reasoner is then only used for the remaining parts of the ontology. Note, our approach works more from the opposite direction: we apply the saturation and simply ignore (or partially process) unsupported features and, then, we detect which parts are not completely handled. Since MORe uses other reasoners as black-boxes, it is, in principle, possible to combine arbitrary reasoning procedures by adapting the module extraction. However, as of now, all fully-fledged OWL 2 reasoners are based on variants of tableau calculi and the efficient reasoning systems for interesting fragments are usually using variants of saturation procedures (e.g., completion- and consequence-based reasoning), whereby the combination of tableau and saturation algorithms currently seems to be the only interesting one.

Due to the black-box approach, the technique realised in MORe is very flexible. For example, it is easily possible to exchange the fully-fledged reasoner with a reasoning system for which it is known that it works best for certain kinds of ontologies. Our approach, on the other hand, has to be implemented into one single reasoning system and requires the support of certain techniques, such as binary absorption, to work well. Moreover, compatible data structures have to be used for both kinds of procedures, which usually means that an appropriate saturation algorithm has to be integrated into a tableau-based reasoning system. Our approach, however, has also various advantages. For example, our saturation uses the same representation of ontologies as tableau algorithms and, therefore, the ontology has to be loaded only once. In contrast, the reasoners used by MORe have to separately load the ontology (or parts thereof) since they are used as black-boxes and, usually, they also do not have compatible data structures. Furthermore, our approach is much more tolerant for the usage of features outside the efficiently supported fragment. Some of our optimisations can also be used when all saturated nodes are critical, which could, for example, be the case if the ontology contains non-absorbable GCIs. In addition, we have presented an extension that allows for fixing critical parts in the saturation, whereby unsupported features are not problematic if they are only rarely used in the ontology. In contrast, MORe has to reduce the module for the efficient reasoner as long as the module contains unsupported features. Thus, our approach promises a better pay-as-you-go behaviour. Moreover, we can use intermediate results from the saturation, whereas the technique in MORe relies on the

externally provided interfaces of the reasoners, which usually only provides basic information such as the satisfiability of concepts and the subsumers of classes. Therefore, our integration of the saturation procedure obviously allows for more sophisticated optimisation techniques such as the transfer of inferred consequences and the blocking of the processing with the tableau algorithm.

Although both approaches are in principle applicable to different reasoning tasks, our technique automatically improves reasoning as long as the reasoning task is reduced to consistency checking with the tableau algorithm. For example, in order to support the satisfiability testing of complex concepts, our approach does not need any adaptations. For MORe, however, it would be necessary to check whether the complex concept is in the module that can be handled by the efficient reasoner in order to achieve an improvement. Last but not least, we do not need the module extraction technique in our approach, which can also take a significant amount of time. This is especially an advantage for ontologies that are almost completely in the efficiently supported fragment since our approach does not have a similar overhead as the module extraction for such ontologies.

Another reasoning system that combines different reasoning techniques is WSReasoner (Song et al., 2012), which uses a weakening and strengthening approach for the classification of ontologies. To be more precise, the ontology is first rewritten into a simpler one (the weakening of the ontology), where not supported language features are (partially) expressed in the fragment that can be handled by the efficient reasoner. Then, a strengthened version of the weakened ontology is created, where axioms are added such that at least also the consequences of the original ontology are implied. The weakened and the strengthened ontologies are then classified by the specialised reasoner and possible differences in the obtained subsumtion relations are verified with a fully-fledged reasoner. Also for WSReasoner, the fragment specific reasoner (usually based on a saturation procedure) and the fully-fledged reasoner (usually based on a tableau calculus) are used as black-boxes, which makes them, in principle, exchangeable. However, the weakening and strengthening also has to be adapted to the language fragment of the efficient reasoner.

Although the technique of WSReasoner is different to the one of MORe, the advantages and disadvantages in comparison with our approach are in principle the same. However, the approach of WSReasoner is not as easily extendible to more language features and, as of now, it is only presented for the DL $\mathcal{ALCHIO}$ (with the elimination/encoding of transitive roles also for $\mathcal{SHIO}$). Moreover, since the nominals are simplified to fresh atomic concepts, the approach cannot straightforwardly be used for all reasoning tasks. If such a simplification is, however, applicable, then it often improves the reasoning performance for corresponding ontologies.

Similarly to WSReasoner, PAGOdA (Zhou et al., 2014, 2015) also uses the weakening and strengthening approach, however, for different reasoning tasks and by delegating a different fragment of the ontology to a specialised reasoner. In particular, PAGOdA is designed for ABox reasoning (e.g., conjunctive query answering) and delegates the majority of the computational workload to an efficient datalog reasoner. If the lower bound from the datalog reasoner does not match with the upper bound, then PAGOdA delegates the query with the relevant parts of the ABox to a fully-fledged reasoner. In order to keep the relevant parts as small as possible, PAGOdA uses additional optimisations such as relevant subset extraction, summarisation, and dependency analysis. However, these additional

optimisations also carry the risk that every use of the fully-fledged reasoner introduces additional overhead, which could be problematic for ontologies where a lot of work has still to be done by the fully-fledged reasoner. Moreover, maintaining several naive representations of the entire ABox can easily multiply memory requirements.

7. Implementation and Evaluation

We extended Konclude[3] (Steigmiller, Liebig, & Glimm, 2014) with the saturation procedure shown in Section 3 and with the optimisations presented in Section 4 and 5. Konclude is a tableau-based reasoner for $\mathcal{SROIQ}$ (Horrocks et al., 2006) with extensions for handling nominal schemas (Steigmiller et al., 2013). Konclude integrates many state-of-the-art optimisations such as lazy unfolding, dependency directed backtracking, caching, etc. Moreover, Konclude uses partial absorption (Steigmiller et al., 2014b) in order to significantly reduce the non-determinism in ontologies and, therefore, Konclude is well-suited for the proposed coupling with saturation procedures.

Our integration of the saturation in Konclude completely covers the language features of the DL Horn-$\mathcal{SHIF}$ by using the saturation extensions described in Section 5.1, where universal restrictions that propagate concepts to successors and the merging of successors/predecessors due to functional at-most restrictions are handled. 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, Konclude's saturation procedure only supports a very limited handling of ABox data. This is due to a design decision with which we try to avoid several representations of individuals (and derived consequences for these) in the reasoning system. Since the saturation could easily be incomplete for the ABox (e.g., since disjunctions are asserted to few individuals or due to datatypes), the ABox has often also be handled by the tableau algorithm and several representations of the ABox can multiply memory requirements. Hence, Konclude primarily handles ABox 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 addition, Konclude saturates the concepts that might be required for a certain reasoning task upfront in a batch processing mode, whereby the switches between the tableau and the saturation algorithm can be reduced significantly. Moreover, we sort the concepts that occur in the knowledge base and saturate them in a specific order to maximise the amount of data that can be shared between the saturated nodes. For example, if the knowledge base contains the axiom $A \sqsubseteq B$, then we first saturate B and we use the data from v_B to initiate v_A. In particular, by copying the node labels, many rule applications can be skipped, which significantly improves the performance of the saturation procedure. Furthermore, this also reduces the effort for the saturation status detection. For instance, if v_B does not satisfy an at-most restriction, then this at-most restriction is also not satisfied for v_A.

In the following, we present a detailed evaluation that shows the effects of Konclude's integrated saturation procedure and the presented optimisations for the support of the fully-fledged tableau algorithm. In addition, we compare the reasoning times of Konclude with the ones of other state-of-the-art reasoners that support TBox reasoning for (almost) all features

3. Konclude is freely available at `http://www.konclude.com/`

Repository	# Ontologies	Axioms Ø	Axioms $Q_{0.5}$	Classes Ø	Classes $Q_{0.5}$	Properties Ø	Properties $Q_{0.5}$	Individuals Ø	Individuals $Q_{0.5}$
Gardiner	292	5,842	96	1,788	14	44	7	85	2
NCBO BioPortal	403	27,180	1,116	7,518	339	48	13	1,766	0
NCIt	185	178,818	167,667	69,720	68,862	116	123	0	0
OBO Foundry	502	37,349	1,292	6,753	509	24	4	20,905	6
Oxford	394	73,921	3,433	8,543	500	53	11	18,273	5
TONES	203	7,707	352	2,864	96	40	5	65	0
Google crawl	414	6,869	255	1,127	39	102	44	824	1
OntoCrawler	548	2,574	136	124	17	93	20	633	0
OntoJCrawl	1,696	6,281	311	1,772	50	62	9	838	0
Swoogle crawl	1,638	2,778	132	416	21	36	10	879	0
ORE2014 dataset	16,555	16,017	594	3,846	87	101	53	1,801	50
ALL	22,830	16,647	594	4,020	74	97	50	2,271	29

Table 5: Statistics of ontology metrics for the evaluated ontology repositories (Ø stands for average and $Q_{0.5}$ for median)

of the DLs $\mathcal{SROIQ}$, namely FaCT++ 1.6.3 (Tsarkov & Horrocks, 2006), HermiT 1.3.8 (Glimm, Horrocks, Motik, Stoilos, & Wang, 2014), MORe 0.1.6 (Armas Romero et al., 2012), and Pellet 2.3.1 (Sirin, Parsia, Cuenca Grau, Kalyanpur, & Katz, 2007). The evaluation uses a large test corpus of ontologies,[4] which has been obtained by collecting all downloadable and parsable ontologies from

- the Gardiner ontology suite (Gardiner, Horrocks, & Tsarkov, 2006),
- the NCBO BioPortal (Whetzel et al., 2011),
- the National Cancer Institute thesaurus (NCIt) archive (National Cancer Institute, 2003),
- the Open Biological Ontologies (OBO) Foundry (Smith et al., 2007),
- the Oxford ontology library (Information Systems Group, 2012),[5]
- the TONES repository (Information Management Group, 2008),
- the subsets of the OWLCorpus (Matentzoglu, Bail, & Parsia, 2013) that were gathered by the crawlers Google, OntoCrawler, OntoJCrawl, and Swoogle,[6] and
- the ORE2014 dataset (Matentzoglu & Parsia, 2014).

4. The test corpus and the evaluated version(s) of Konclude v0.6.1 can be found online at `http://www.derivo.de/en/products/konclude/paper-support-pages/tableau-saturation-coupling.html`
5. Note that the Oxford ontology library also contains other repositories (e.g., the Gardiner ontology suite), which we ignored in order to avoid too much redundancy.
6. 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.

Ontology	Expressiveness	Axioms	Classes	Properties	Individuals
Gazetteer	$\mathcal{ALE}+$	1,170,573	518,196	16	1
EL-GALEN	$\mathcal{ALEH}+$	60,633	23,136	950	0
Full-GALEN	$\mathcal{ALEHIF}+$	61,782	23,136	950	0
Biomodels	$\mathcal{SRIF}$	847,794	187,520	70	220,948
Cell Cycle v2.01	$\mathcal{SRI}$	731,482	106,398	469	0
NCI v06.12d	$\mathcal{ALCH}$	141,957	58,771	124	0
NCI v12.11d	$\mathcal{SH}$	229,713	95,701	110	0
SCT-SEP	$\mathcal{SH}$	109,959	54,974	9	0
FMA v2.0-CNS	$\mathcal{ALCOIF}$	165,000	41,648	148	85
OBI	$\mathcal{SHOIN}$	32,157	3,533	84	160

Table 6: Ontology metrics for selected benchmark ontologies

Note that the ORE2014 dataset is a collection of ontologies from several sources and it redundantly contains many of the ontologies that are also contained by the other repositories. However, many ontologies of the ORE2014 dataset have been adapted and approximated to fit the requirements of certain OWL 2 profiles (e.g., by removing datatypes that are not in the OWL 2 datatype map, by enforcing a regular role hierarchy, and by adding declarations for undeclared entities). We used the OWL API for parsing and we converted all ontologies to self-contained OWL/XML files, where we created, for each of the 1,380 ontologies with imports, a version with resolved imports and another version, where the import directives are simply removed (which allows for testing the reasoning performance on the main ontology content without imports, which are frequently shared by many ontologies). Table 5 shows an overview of our obtained test corpus with overall 22,830 ontologies including statistics of ontology metrics for the source repositories.

In addition to our test corpus, we present results for explicitly selected ontologies (shown in Table 6) which are frequently used in many evaluations. This allows for directly showing the effects of our approach for well-known benchmark ontologies and enables a more concrete comparison. Note that Table 6 is separated into EL (upper part) and non-EL ontolgies (lower part). As EL ontologies, we chose the well-known Gazetteer and EL-GALEN, where the latter one is obtained by removing functionality and inverses from the Full-GALEN ontology, which we also selected for benchmarking. In addition, we evaluated Biomodels and Cell Cycle v2.01, which are large but mainly deterministic ontologies from the NCBO BioPortal, NCI v06.12d and NCI v12.11d, which are different versions of the NCI-Thesaurus ontology from the NCIt archive, SCT-SEP, which denotes the SNOMED CT anatomical model ontology (Kazakov, 2010), FMA v2.0-CNS, which is a version of the Foundational Model of Anatomy (Golbreich, Zhang, & Bodenreider, 2006), and OBI, which represents a recent version of the Ontology for Biomedical Investigations (Brinkman et al., 2010).

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 144 GB RAM under a 64bit Ubuntu 12.04.2 LTS. Our evaluation focuses on classification, which is a central reasoning task supported by many reasoners and, thus, it is ideal for comparing results. In principle, we only measured the classification time, i.e., the time spent for parsing and loading ontologies as well as writing classification output to files is not included for the pre-

sented results. This is an advantage for reasoners that already perform some preprocessing while loading, which is, however, not the case for Konclude since Konclude uses a lazy processing approach where also the preprocessing is triggered with the classification request. This also seems to be confirmed by the accumulated loading times over all ontologies in the evaluated repositories, which are 6,304 s for Konclude, 10,877 s for MORe, 13,210 s for FaCT++, 22,458 s for Pellet, and 61,293 s for HermiT. Note that HermiT directly clausifies the ontologies while loading (i.e., it converts the axioms into HermiT's internal representation based on DL-clauses), which can easily take a lot of time if ontologies intensively use cardinality restrictions. We also ignored all errors that were reported by (other) reasoners, i.e., if a reasoner stopped the processing of an ontology (e.g., due to unsupported axioms or program crashes), then we only measured the actual processing time. This is also a disadvantage for Konclude since Konclude processed all ontologies (however, Konclude also ignored parts of role inclusion axioms if they were not regular as specified by OWL 2 DL). In contrast, MORe reported errors for 803, FaCT++ for 944, Pellet for 1,285, and HermiT for 1,483 ontologies in our corpus. The reasoners often cancelled the processing due to unsupported or malformed datatypes. Another frequently reported error consisted of different individual axioms for which only one individual was specified. In addition, HermiT completely refused processing ontologies with irregular role inclusion axioms (which are, however, only rarely present in our test corpus).

For the evaluation of the ontology repositories, we used the time limit of 5 minutes. For the selected benchmark ontologies, we cancelled the classification task after 15 minutes since these ontologies are relatively large. Moreover, we averaged the results for the selected benchmark ontologies over 3 separate runs, which was not necessary for the evaluated repositories since the large amount of ontologies automatically compensates the non-deterministic behaviours of the reasoners, i.e., the accumulated (classification) times for separate runs over many ontologies are almost identical. Although some reasoners support parallelisation, we configured all reasoners to use only one worker thread, which allows for a comparison independently of the number of CPU cores and facilitates the presentation of the improvements through saturation.

7.1 Evaluation of Saturation Optimisations

The presented optimisations are integrated in Konclude in such a way that they can separately be activated and deactivated. Hence, we can evaluate and compare the performance improvements for the different optimisations. Please note that deactivating optimisations in Konclude can cause disproportionate performance losses since appropriate replacement optimisations, which could compensate the deactivated techniques to some extent, are often not integrated in Konclude. For example, many reasoning systems use the completely defined concepts optimisation (Tsarkov & Horrocks, 2005) to identify those classes of an ontology for which all subsumption relations can directly be extracted from the ontology axioms and, thus, satisfiability and subsumption tests are not necessary to correctly insert these classes into the class hierarchy. Clearly, such an optimisation is not necessary for Konclude, because we can extract all subsumers of a class from the saturation if the saturated representative node is not critical. Hence, the performance with deactivated optimisations

might be worse than it has to be. Nevertheless, we evaluated the versions of Konclude, where

- all saturation optimisations are activated (denoted by ALL), and
- none of the saturation optimisations are activated (denoted by NONE),

in combination with the activation/deactivation (denoted by $+/-$) of the following modifications:

- RT (standing for **r**esult **t**ransfer), where the transfer of (possibly intermediate) results from the saturation into completion graphs (as presented in Section 4.1) is activated/deactivated. More precisely, we initialise new nodes in a completion graph with the consequences available in the saturation graph and we block the processing of (successor) nodes as long as they are identically labelled as the non-critical nodes in the (deterministic or non-deterministic) saturation graph. As described in Section 5.2, nominal dependent nodes are handled by reactivating the processing if the nodes for the dependent nominals become modified in the completion graph. For this, we implemented an exact tracking of nominal dependent nodes in the completion graph as well as in the saturation graph.

- SE (standing for **s**ubsumer **e**xtraction), where the extraction of subsumers from the saturation (as presented in Section 4.2) is activated/deactivated. If the representative nodes for atomic concepts are not critical, then Konclude use them to extract all subsumers (besides completely defined concepts) and, otherwise, the derived atomic concepts are only used as told subsumers. Note that, if completely defined concepts are not in a node label, then the candidate concepts are interpreted as if the corresponding completely defined concepts are non-deterministically derived, i.e., as possible subsumers. If the SE optimisation is deactivated, then Konclude extracts some simple told subsumers from the axioms of the knowledge base in order to initialise the classification algorithm, which is also in Konclude based on the known and possible sets classification (Glimm et al., 2012).

- MM (standing for **m**odel **m**erging), where the model merging with the saturation graph (as presented in Section 4.3) is activated/deactivated. The candidate concepts are obtained in Konclude with the partial absorption technique (Steigmiller et al., 2014b) and the model merging is only applied for the first initialisation of the known and possible subsumers of an atomic concept. In particular, we avoid a repeated model merging for the same possible subsumption relation on different nodes (in possibly different completion graphs) since this could result in a significant overhead while possibly only few new non-subsumptions are identified.

- ES (standing for **e**xtended **s**aturation), where the handling of universal restrictions and of functional at-most restrictions for successors in the saturation (as presented in Section 5.1) is activated/deactivated. Note that the integrated saturation procedure becomes complete for Horn-$\mathcal{SHIF}$ knowledge bases if this optimisation is activated, whereas completeness is only guaranteed for $\mathcal{ELH}$ knowledge bases if it is deactivated.

- PS (standing for **p**atched **s**aturation), where the patching of the saturation graph with data from completion graphs (as presented in Section 5.2) is activated/deactivated. To ensure patch compatibility for nominal dependent nodes, we use the completion graph caching technique integrated in Konclude (Steigmiller et al., 2015) such that only those nominal nodes are identified for which possibly different consequences are derived as in the initial completion graph. Since Konclude supports an exact tracking of nominal dependency in the completion graph, we save the dependent nominals in patches and propagate them in the saturation graphs such that the processing of a node only has to be reactivated if a node for a dependent nominal becomes modified. Konclude further incorporates an exact tracking of which facts are the causes of which derived facts (Steigmiller, Liebig, & Glimm, 2012) and this is used to also extract patches from non-root nodes (as also sketched in Section 5.2). Moreover, if it can be discovered that a satisfiability test for a concept does not result in a fully expanded and clash-free completion graph, i.e., the concept is unsatisfiable, then Konclude patches the saturation graphs with the $\bot$-concept such that other unsatisfiable concepts are also revealed.

For example, NONE+MM denotes the version of Konclude, where all saturation optimisations except the model merging with the saturation graph are deactivated.

Based on the version NONE, Table 7 shows the performance improvements for the activation of the saturation optimisations RT, SE, and MM. In addition, the results for ALL are shown, where all optimisations are activated simultaneously. Please note that ES and PS are optimisations to further improve the saturation procedure and, therefore, their evaluation only makes sense in combination with other saturation optimisations. The most significant improvements are achieved by the transfer of saturation results into completion graphs (RT), which often reduces the effort for the tableau algorithm significantly. The model merging optimisation (MM) primarily improves the classification performance for the NCI-Thesaurus ontologies in the NCIt archive, but does not have a similar significant impact for the other repositories. Since many NCI-Thesaurus ontologies contain complete definitions of the form $A \equiv \exists r.B_1 \sqcap \forall s.B_2$, the model merging with the candidate concepts (as demonstrated in Section 4.3) allows for pruning many subsumptions while performing satisfiability tests for the atomic concepts. There are also improvements through the extraction of subsumers from the saturation (SE), but, compared to the improvements of the other optimisations, they are only significantly better for the NCBO BioPortal. In particular, the NCBO BioPortal contains many large but relatively simple ontologies that can almost be completely handled by the saturation and, therefore, it is not necessary to perform satisfiability tests for every class with the tableau algorithm if the SE optimisation is activated in order to determine the (possible) subsumers. Nevertheless, if all saturation optimisations are activated, then we are often able to achieve much larger performance improvements for almost all repositories. On the one hand, this is caused by the additionally activated ES and PS optimisations, but on the other hand, the reasoning system can utilise several synergy effects from the saturation (obviously, the concepts have to be saturated only once for all optimisations).

Table 8 analogously shows the performance improvements by activating the saturation optimisations RT, SE, and MM for the selected benchmark ontologies. The saturation optimisations can significantly improve the classification performance for several ontologies.

Repository	NONE	NONE+RT	NONE+SE	NONE+MM	ALL
Gardiner	526	508	414	490	108
NCBO BioPortal	2,259	2,039	580	2,326	260
NCIt	28,603	28,434	27,940	3,163	1,942
OBO Foundry	3,020	812	877	2,829	748
Oxford	7,976	4,639	5,866	8,013	2,484
TONES	1,734	1,481	1,568	756	250
Google crawl	798	463	670	794	112
OntoCrawler	27	29	30	31	27
OntoJCrawl	3,405	1,166	2,232	2,504	715
Swoogle crawl	3,477	2,670	2,820	2,283	1,187
ORE2014 dataset	115,494	80,673	98,630	115,232	29,841
ALL	167,320	122,914	141,628	138,421	37,674

Table 7: Accumulated classification times (in seconds) with separately activated saturation optimisations for the evaluated ontology repositories

Ontology	NONE	NONE+RT	NONE+SE	NONE+MM	ALL
Gazetteer	34.8	30.1	14.0	37.9	13.3
EL-GALEN	761.0	5.5	1.6	762.6	1.4
Full-GALEN	$\geq$ 900.0	$\geq$ 900.0	$\geq$ 900.0	$\geq$ 900.0	12.0
Biomodels	241.5	50.6	18.2	148.7	16.2
Cell Cycle v2.01	$\geq$ 900.0	$\geq$ 900.0	7.6	$\geq$ 900.0	7.2
NCI v06.12d	$\geq$ 900.0	$\geq$ 900.0	$\geq$ 900.0	17.9	13.9
NCI v12.11d	17.7	14.6	8.8	16.0	8.2
SCT-SEP	$\geq$ 900.0	339.8	279.4	383.1	173.1
FMA v2.0-CNS	$\geq$ 900.0	$\geq$ 900.0	$\geq$ 900.0	$\geq$ 900.0	72.7
OBI	1.3	0.8	0.7	2.1	0.6

Table 8: Classification times (in seconds) with separately activated saturation optimisations for the evaluated benchmark ontologies

In particular, with the optimisations, Konclude can handle all ontologies in a reasonable amount of time, whereas Konclude timed out for five of these ontologies if the saturation optimisations were not used. Very difficult ontologies such as Full-GALEN and FMA v2.0-CNS can only be handled if more sophisticated saturation optimisations are used (e.g., SE, PS). It can also be observed that, for many ontologies, only specific optimisations are crucial, which is, however, also not very surprising. For example, it is clear that the MM optimisation cannot improve the performance for deterministic ontologies since they do not have possible subsumers for which the model merging could be applied.

Table 9 shows the performance changes for the separate deactivation of saturation optimisations based on the ALL configuration. The evaluation of optimisations is also very interesting from this perspective, because the saturation of many concepts can easily require a significant amount of reasoning time and, by separately deactivating single optimisations, the overhead of the saturation is not only associated with a separately activated optimisa-

Repository	ALL	ALL−RT	ALL−SE	ALL−MM	ALL−ES	ALL−PS
Gardiner	108	134	201	90	396	106
NCBO BioPortal	260	624	1,980	618	582	709
NCIt	1,942	2,041	2,580	27,952	2,000	1,960
OBO Foundry	748	1,052	453	473	774	453
Oxford	2,484	3,987	3,701	2,398	3,658	2,537
TONES	250	143	366	1,377	226	633
Google crawl	112	790	731	706	412	733
OntoCrawler	27	30	62	30	30	35
OntoJCrawl	715	2,017	1,445	702	764	879
Swoogle crawl	1,187	1,427	1,209	2,456	1,348	1,201
ORE2014 dataset	29,841	56,128	56,469	37,760	51,400	61,036
ALL	37,674	68,374	69,286	71,562	61,590	70,281

Table 9: Accumulated classification times (in seconds) with separately deactivated saturation optimisations for the evaluated ontology repositories

Ontology	ALL	ALL−RT	ALL−SE	ALL−MM	ALL−ES	ALL−PS
Gazetteer	13.3	13.6	27.9	13.2	13.7	13.5
EL-GALEN	1.4	1.5	4.8	1.4	1.5	1.5
Full-GALEN	12.0	12.7	25.1	11.8	$\geq$ 900.0	12.7
Biomodels	16.2	17.2	47.1	15.6	16.2	16.6
Cell Cycle v2.01	7.2	7.5	$\geq$ 900.0	7.1	7.3	6.9
NCI v06.12d	13.9	15.0	15.7	$\geq$ 900.0	13.8	13.2
NCI v12.11d	8.2	8.7	13.0	7.6	7.4	7.7
SCT-SEP	173.1	280.4	337.1	161.9	167.5	168.7
FMA v2.0-CNS	72.7	60.3	28.2	180.3	66.6	$\geq$ 900.0
OBI	0.6	0.8	0.8	0.7	0.7	0.7

Table 10: Classification times (in seconds) with separately deactivated saturation optimisations for selected benchmark ontologies

tion. Furthermore, this allows for evaluating whether some optimisations are superfluous and which effects are caused by the saturation improvements ES and PS, which are only useful in combination with other saturation optimisations. Table 9 reveals that some saturation optimisations are completely irrelevant for some repositories. Moreover, the deactivation of optimisations can also improve the performance for several repositories, e.g., the deactivation of RT results in better reasoning times for the ontologies in the TONES repository and the deactivation of MM causes some minor performance improvements for the OntoJCrawl ontologies. However, by considering all repositories, each optimisation is indeed justified. In particular, if any of the presented saturation optimisations is deactivated, then the reasoning times increase by at least 55 %. This is also caused by several difficult ontologies in the ORE2014 dataset, such as variants of the KB_Bio_101 ontology (Chaudhri, Wessel, & Heymans, 2013), which can only be handled by Konclude if almost all saturation optimisations are used. The patching of the saturation graph (PS) with the data from the initial

consistency test is often required for the complete/sufficient handling of nominals within the saturation procedure, the saturation extensions (ES) enable a primitive handling of (qualified) cardinality restrictions even for very big cardinalities (due to the reuse of nodes in the saturation graph), and the result transfer (RT) as well as the subsumer extraction (SE) reduce or avoid the work for the tableau algorithm, which is particularly useful for very big and highly cyclic ontologies. Although the MM optimisation is similar important for the ORE2014 dataset, it is the only optimisation that significantly reduces the effort of Konclude for the NCI-Thesaurus ontologies from the NCIt archive.

The performance changes for the separate deactivation of saturation optimisations for the evaluated benchmark ontologies are depicted in Table 10. Again, it can be observed that often only specific optimisations are important for the ontologies. For example, only the deactivation of the SE optimisations significantly decreases the performances for the Biomodels and Cell Cycle v2.01 ontologies. Since Full-GALEN is highly cyclic and has many consequences that are caused by functional cardinality restrictions as well as inverse roles, the tableau algorithm has difficulties to find appropriate blocker nodes in the completion graph and, therefore, it can only be handled if the saturation is extended to these language features (as realised by the ES optimisation). In contrast, FMA v2.0-CNS has many unsatisfiable classes and, as soon as the tableau algorithm can find such an unsatisfiable class, the saturation graph can be patched (realised with the PS optimisation) and the $\bot$-concept can directly be propagated to many other classes, whereby many satisfiability tests with the tableau algorithm become unnecessary.

7.2 Evaluation of the Saturation Effort

Table 11 shows the distribution of the processing times w.r.t. Konclude's processing stages for the classification of the evaluated repositories with the version ALL. Unsurprisingly, the majority of the processing time (61.5 %) is spent for the classification process itself. In contrast, the saturation of all those concepts that are potentially required for the classification requires only 12.1 % together with the detection of the saturation status. The latter one can, however, usually be neglected in terms of processing time since our implementation is very efficient. For example, if a node is detected as critical, then the criticality status is immediately propagated to all dependent nodes and, as a consequence, they do not have to be tested. Moreover, we use a criticality testing queue that is filled during saturation if concepts are added to node labels that potentially influence the criticality status. Hence, the status detection does not have to iterate through all node labels. Although it is in principle possible to design ontologies where the saturation can be relatively inefficient (in particular w.r.t. the memory requirements), such ontologies hardly occur in practice. In particular, with the data sharing for node labels that is realised in Konclude, the saturation does not cause significant problems for the evaluated repositories, which is also reflected by the short processing time for the saturation stage. Consistency checking can usually also be performed efficiently, but several evaluated repositories (e.g., the Swoogle crawl) also contain very difficult ontologies for which the tableau algorithm cannot find a fully expanded and clash-free completion graph within the time limit. Building the internal representation as well as preprocessing are also realised very efficiently in Konclude and do not cause problems for the evaluated repositories.

Repository	Building	Preprocessing	Saturation	Consistency	Classification
Gardiner	10.8	30.6	32.5	1.8	24.3
NCBO BioPortal	27.4	17.1	23.6	3.1	28.8
NCIt	7.5	11.0	11.2	1.9	68.4
OBO Foundry	16.3	5.0	6.7	46.6	25.4
Oxford	5.6	10.6	12.2	20.3	51.3
TONES	2.6	3.3	5.6	0.6	87.9
Google crawl	11.6	7.9	19.4	8.3	52.9
OntoCrawler	38.3	10.7	17.5	21.1	12.5
OntoJCrawl	8.2	4.6	4.4	48.1	34.7
Swoogle crawl	2.2	1.0	1.8	26.7	68.2
ORE2014 dataset	4.8	5.8	12.5	13.4	63.4
ALL	5.4	6.3	12.1	14.6	61.5

Table 11: Distribution of processing time w.r.t. different processing stages (in %)

7.3 Comparison with other Approaches

As mentioned in Section 6, there exist other approaches that also use saturation-based reasoning techniques to improve fully-fledged tableau algorithms. For example, MORe uses module extraction to delegate as much work as possible to an efficient reasoner that is specialised for a specific fragment in order to classify ontologies. Since an early development version of MORe is available, we evaluated MORe with our test corpus and we compare the results to our approach in the following. We used MORe in combination with ELK 0.4.1 (Kazakov, Krötzsch, & Simančík, 2014) and HermiT 1.3.8, but other combinations are also possible since these reasoners are used as black-boxes.

The left-hand side of Table 12 shows the accumulated classification times (in seconds) for the versions of Konclude where all saturation optimisations are deactivated (version NONE in Column 2) and all saturation optimisations are activated (version ALL in Column 3) for the different repositories. Furthermore, the improvement from the version NONE to the version ALL is given in percent (Column 4 of Table 12). For example, by using all saturation optimisations presented here, the accumulated reasoning time for all repositories is reduced by 77.5 % for Konclude. On the right-hand side of Table 12, we have analogously depicted the accumulated reasoning times for HermiT (Column 5) and MORe (Column 6), and also the percentage of HermiT's reasoning time that can be reduced by MORe (Column 7).

Note, the accumulated loading times for all repositories are 61,293 s for HermiT and 10,877 s for MORe, where the difference of 50,416 s can be explained by the additional preprocessing that is directly performed in HermiT's loading stage, whereas MORe starts the processing of the ontologies not before the classification request. Of course, MORe also uses HermiT internally to process parts of ontologies that cannot be handled by the OWL 2 EL reasoner ELK, but the required time for loading these parts with HermiT is then counted as reasoning/classification time for MORe. Hence, we made the comparison fair by adding the additional preprocessing time in the loading stage to HermiT's classification time, i.e., the shown classification times for HermiT are extended by the difference between the loading times of HermiT and MORe.

Repository	NONE [s]	ALL [s]	↓ [%]	HermiT [s]	MORe [s]	↓ [%]
Gardiner	526	108	79.5	1,773	1,537	13.3
NCBO BioPortal	2,259	260	88.5	5,901	4,187	29.0
NCIt	28,603	1,942	93.2	26,435	26,600	−0.6
OBO Foundry	3,020	748	75.2	6,654	4,474	32.8
Oxford	7,976	2,484	68.9	12,865	8,083	37.2
TONES	1,734	250	85.6	2,342	2,184	6.7
Google crawl	798	112	86.0	1,917	1,629	15.0
OntoCrawler	27	27	0.0	1,863	893	52.1
OntoJCrawl	3,405	715	79.0	8,555	4,546	46.9
Swoogle crawl	3,477	1,187	65.9	4,857	4,270	12.1
ORE2014 dataset	115,494	29,841	74.2	294,124	166,589	43.9
ALL	167,320	37,674	77.5	367,283	224,982	38.7

Table 12: Comparison of improvements through saturation between the approaches realised in Konclude and MORe for the accumulated classification times of the evaluated ontology repositories (in seconds and %)

Ontology	NONE [s]	ALL [s]	↓ [%]	HermiT [s]	MORe [s]	↓ [%]
Gazetteer	34.8	13.3	51.2	≥ 900.0	18.2	≥ 98.0
EL-GALEN	761.0	1.4	98.0	≥ 900.0	2.6	≥ 99.7
Full-GALEN	≥ 900.0	12.0	≥ 98.7	≥ 900.0	≥ 900.0	–
Biomodels	241.5	16.2	93.3	788.8	648.8	17.7
Cell Cycle v2.01	≥ 900.0	7.2	≥ 99.2	≥ 900.0	≥ 900.0	–
NCI v06.12d	≥ 900.0	13.9	≥ 98.5	211.9	208.0	1.9
NCI v12.11d	17.7	8.2	53.8	92.7	83.3	10.1
SCT-SEP	≥ 900.0	173.1	≥ 80.1	≥ 900.0	≥ 900.0	–
FMA v2.0-CNS	≥ 900.0	72.7	≥ 75.9	≥ 900.0	≥ 900.0	–
OBI	1.3	0.6	53.8	32.5	2.3[7]	93.0

Table 13: Improvements through saturation between the approaches in Konclude and MORe for the classification times of selected benchmark ontologies (in seconds and %)

Table 12 reveals that MORe can significantly improve the reasoning time of HermiT for almost all repositories. In particular, MORe saves 52.1 % of HermiT's classification time for the ontologies from OntoCrawler. Nevertheless, there are still many ontologies in these repositories, where MORe is not able to reduce the effort of HermiT such that they can be classified within the time limit (HermiT timed out for 786 and MORe for 590 ontologies, respectively). In contrast, Konclude integrates a more sophisticated interaction between the tableau algorithm and the saturation procedure and, therefore, the improvements by the saturation optimisations are significantly better for many repositories. As a result, the

7. The class hierarchy computed by MORe for OBI does not coincide with the results of HermiT and Konclude.

ALL version of Konclude reached the time limit only for 69 ontologies. Note, already the version NONE of Konclude, where all saturation optimisations are deactivated, outperforms HermiT and MORe for many ontologies, which is probably due to the difference in the integrated optimisations. For example, Konclude uses several caching techniques to save and reuse intermediate results, which usually improves the reasoning performance a lot. Moreover, the partial absorption (Steigmiller et al., 2014b) integrated in Konclude significantly reduces non-determinism also for very expressive ontologies. Also note that MORe does not yet completely support all language features of $\mathcal{SROIQ}$ and, therefore, it does not always output the correct class hierarchy (for the evaluated ontology repositories, 1,441 class hierarchies computed by MORe do not coincide with the results of HermiT and Konclude).

Table 13 analogously shows the performance improvements for the selected benchmark ontologies. Again, it can be observed that the improvements through the saturation are often better for Konclude than for MORe, especially if ontologies are considered for which the version NONE of Konclude still requires a lot of reasoning time.

7.4 Comparison with State-of-the-Art Reasoners

We also evaluated the classification times for the state-of-the-art reasoners FaCT++ and Pellet, which are compared with the other reasoners HermiT, Konclude, and MORe in Table 14. Note, Table 14 only shows the accumulated classification times that are actually reported by the reasoners, i.e., we did not compensate differences in the loading times.

It can be observed that Konclude outperforms all other reasoners for all evaluated repositories, which is mainly due to the integrated saturation optimisations. FaCT++ is the only reasoner that can efficiently handle the majority of all NCI-Thesaurus ontologies in the NCIt archive also without saturation optimisations. Nevertheless, the model merging with the saturation graph allows for pruning many possible subsumers in Konclude, whereby the classification performance can further be improved and, therefore, Konclude is able to outperform FaCT++ also for the NCIt archive. Considering all repositories, Konclude produced the fewest timeouts (69), followed by MORe (590), HermiT (786), FaCT++ (822), and Pellet (1,470).

Nevertheless, there are a few ontologies for which Konclude's performance is not optimal and for which other reasoners are sometimes able to outperform Konclude. For example, Konclude requires 54.6 s for the classification of the atom-complex-proton-2.0 ontology from the TONES repository, whereas Pellet only requires 11.4 s (FaCT++ and HermiT timed out). In particular, the handling of (large) cardinalities can easily cause problems since, in the worst-case, the tableau algorithm is used to create and merge the corresponding numbers of successor nodes. Although our saturation has a limited handling of at-least cardinality restrictions by using only one representative node as successor, this easily becomes incomplete if also at-most cardinality restrictions are used in ontologies. As a remedy, one could try to extend the saturation procedure to better handle cardinality restrictions or to combine the tableau algorithm with algebraic methods, where cardinality restrictions are handled as a system of linear (in)equations (Haarslev, Sebastiani, & Vescovi, 2011). Moreover, too much non-determinism, e.g., caused by non-absorbable GCIs, can still cause serious issues for tableau-based systems. Examples of such ontologies are variants of enzyo from the Swoogle crawl, which cannot be classified by any of the evaluated reasoners except

Repository	FaCT++	HermiT	Konclude	MORe	Pellet
Gardiner	1,108	1,688	108	1,537	4,006
NCBO BioPortal	5,413	5,570	260	4,187	9,877
NCIt	4,393	25,203	1,942	26,600	17,647
OBO Foundry	7,225	6,258	748	4,474	12,031
Oxford	20,761	12,255	2,484	8,083	27,461
TONES	1,684	1,943	250	2,184	1,755
Google crawl	2,178	1,761	112	1,629	7,496
OntoCrawler	964	1,723	27	893	9,999
OntoJCrawl	11,580	6,757	715	4,546	31,776
Swoogle crawl	2,864	4,073	1,187	4,270	9,212
ORE2014 dataset	241,402	249,637	29,841	166,589	397,794
ALL	299,573	316,867	37,674	224,982	528,891

Table 14: Comparison of accumulated classification times between state-of-the-art reasoners (in seconds) for the evaluated ontology repositories

Ontology	FaCT++	HermiT	Konclude	MORe	Pellet
Gazetteer	$\geq$ 900.0	$\geq$ 900.0	13.3	18.2	480.2
EL-GALEN	$\geq$ 900.0	$\geq$ 900.0	1.4	2.6	135.1
Full-GALEN	$\geq$ 900.0	$\geq$ 900.0	12.0	$\geq$ 900.0	$\geq$ 900.0
Biomodels	2.7[8]	788.8	16.2	648.8	$\geq$ 900.0
Cell Cycle v2.01	$\geq$ 900.0	$\geq$ 900.0	7.2	$\geq$ 900.0	$\geq$ 900.0
NCI v06.12d	13.9	206.1	13.9	208.0	69.6
NCI v12.11d	57.8	78.8	8.2	83.3	306.9
SCT-SEP	$\geq$ 900.0	$\geq$ 900.0	173.1	$\geq$ 900.0	$\geq$ 900.0
FMA v2.0-CNS	$\geq$ 900.0	$\geq$ 900.0	28.3	$\geq$ 900.0	$\geq$ 900.0
OBI	$\geq$ 900.0	31.5	0.6	2.3[9]	$\geq$ 900.0

Table 15: Comparison of classification times between state-of-the-art reasoners (in seconds) for selected benchmark ontologies

for FaCT++ (but also FaCT++ needs a significant amount of time and even reaches the time limit of 5 minutes for some variants) although they have less than 20,000 axioms and only an expressiveness that is ranging from $\mathcal{ALIN}$ to $\mathcal{ALCOIN}$. Also very large $\mathcal{SROIQ}$ ontologies from the Oxford ontology library, such as Mus_musculus consisting of 221,484 axioms, seem currently to be out of reach for existing reasoning systems. Due to their size and complexity, it is even difficult to analyse in which kinds of problems the reasoners are running, but an intensive use of nominals often limits the applicability of optimisation techniques and, hence, often results in poor performance.

Analogously, Table 15 shows the comparison of the classification times between all evaluated reasoners for the selected benchmark ontologies in seconds. Again, with the activated

8. FaCT++ 1.6.3 crashed for the classification of Biomodels after 2.7 seconds.
9. The class hierarchy that is computed by MORe for OBI does not coincide with the results of the other reasoners.

saturation optimisations, Konclude can outperform the other reasoners for almost all ontologies and is able to classify all these benchmark ontologies within the time limit. Compared to the other reasoners, MORe can also achieve good results for several of these ontologies. In particular, MORe only timed out for 4 ontologies, whereas HermiT and Pellet could not classify 6 ontologies in time, and FaCT++ failed for the classification of 7 ontologies. Hence, a support through saturation seems to pay off.

8. Conclusions

In this paper, we have presented a technique for tightly coupling saturation- and tableau-based procedures. Unlike standard completion- and consequence-based saturation procedures, the approach is applicable for 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.

The very good pay-as-you-go behaviour 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. In particular, with these optimisations, Konclude is able to outperform many other state-of-the-art reasoners for a wide range of ontologies often by more than one order of magnitude.

Acknowledgments

The first author acknowledges the support of the doctoral scholarship under the Postgraduate Scholarships Act of the Land of Baden-Wuerttemberg (LGFG). This work was done within the Transregional Collaborative Research Centre SFB/TRR 62 “A Companion-Technology for Cognitive Technical Systems” funded by the German Research Foundation (DFG).

References

Armas Romero, A., Cuenca Grau, B., & Horrocks, I. (2012). MORe: Modular combination of OWL reasoners for ontology classification. In *Proc. 11th Int. Semantic Web Conf. (ISWC'12)*, Vol. 7649 of *LNCS*, pp. 1–16. Springer.

Baader, F., Brandt, S., & Lutz, C. (2005). Pushing the $\mathcal{EL}$ envelope. In *Proc. 19th Int. Joint Conf. on Artificial Intelligence (IJCAI'05)*, pp. 364–369. Professional Book Center.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., & Patel-Schneider, P. (Eds.). (2007). *The Description Logic Handbook: Theory, Implementation, and Applications* (Second edition). Cambridge University Press.

Baader, F., Hollunder, B., Nebel, B., Profitlich, H.-J., & Franconi, E. (1994). An empirical analysis of optimization techniques for terminological representation systems. *J. of Applied Intelligence*, *4*(2), 109–132.

Bate, A., Motik, B., Cuenca Grau, B., Simančík, F., & Horrocks, I. (2015). Extending consequence-based reasoning to $\mathcal{SHIQ}$. In *Proc. 28th Int. Workshop on Description Logics (DL'15)*.

Brinkman, R. R., Courtot, M., Derom, D., Fostel, J., He, Y., Lord, P. W., Malone, J., Parkinson, H. E., Peters, B., Rocca-Serra, P., Ruttenberg, A., Sansone, S., Soldatova, L. N., Jr., C. J. S., Turner, J. A., Zheng, J., & OBI consortium (2010). Modeling biomedical experimental processes with OBI. *J. Biomedical Semantics*, *1*(S-1), S7.

Chaudhri, V. K., Wessel, M. A., & Heymans, S. (2013). KB_Bio_101: A challenge for OWL reasoners. In *Proc. 2nd Int. Workshop on OWL Reasoner Evaluation (ORE'13)*. CEUR.

Gardiner, T., Horrocks, I., & Tsarkov, D. (2006). Automated benchmarking of description logic reasoners. In *Proc. 19th Int. Workshop on Description Logics (DL'06)*, Vol. 198. CEUR.

Glimm, B., Horrocks, I., Motik, B., Shearer, R., & Stoilos, G. (2012). A novel approach to ontology classification. *J. of Web Semantics*, *14*, 84–101.

Glimm, B., Horrocks, I., Motik, B., Stoilos, G., & Wang, Z. (2014). HermiT: An OWL 2 reasoner. *J. of Automated Reasoning*, *53*(3), 1–25.

Golbreich, C., Zhang, S., & Bodenreider, O. (2006). The foundational model of anatomy in OWL: Experience and perspectives. *J. of Web Semantics*, *4*(3), 181–195.

Haarslev, V., Möller, R., & Turhan, A.-Y. (2001). Exploiting pseudo models for TBox and ABox reasoning in expressive description logics. In *Proc. 1st Int. Joint Conf. on Automated Reasoning (IJCAR'01)*, Vol. 2083 of *LNCS*, pp. 61–75. Springer.

Haarslev, V., Sebastiani, R., & Vescovi, M. (2011). Automated reasoning in $\mathcal{ALCQ}$ via SMT. In *Proc. 23rd Int. Conf. on Automated Deduction (CADE'11)*, pp. 283–298. Springer.

Horrocks, I., Kutz, O., & Sattler, U. (2006). 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.

Horrocks, I., & Sattler, U. (1999). A description logic with transitive and inverse roles and role hierarchies. *J. of Logic and Computation*, *9*(3), 385–410.

Horrocks, I., & Sattler, U. (2001). Optimised reasoning for $\mathcal{SHIQ}$. In *Proc. 15th European Conf. on Artificial Intelligence (ECAI'02)*, pp. 277–281. IOS Press.

Horrocks, I., & Sattler, U. (2004). Decidability of $\mathcal{SHIQ}$ with complex role inclusion axioms. *Artificial Intelligence*, *160*(1), 79–104.

Horrocks, I., & Sattler, U. (2007). A tableau decision procedure for $\mathcal{SHOIQ}$. *J. of Automated Resoning*, *39*(3), 249–276.

Horrocks, I., Sattler, U., & Tobies, S. (1999). Practical reasoning for expressive description logics. In *Proc. 6th Int. Conf. on Logic Programming and Automated Reasoning (LPAR'99)*, Vol. 1705 of *LNCS*, pp. 161–180. Springer.

Horrocks, I., Sattler, U., & Tobies, S. (2000). Reasoning with individuals for the description logic $\mathcal{SHIQ}$. In *Proc. 17th Int. Conf. on Automated Deduction (CADE'00)*, Vol. 1831 of *LNCS*, pp. 482–496. Springer.

Horrocks, I., & Tobies, S. (2000). 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.

Hudek, A. K., & Weddell, G. E. (2006). Binary absorption in tableaux-based reasoning for description logics. In *Proc. 19th Int. Workshop on Description Logics (DL'06)*, Vol. 189. CEUR.

Information Management Group (2008). TONES ontology repository. University of Manchester. Available at `http://owl.cs.manchester.ac.uk/repository/`. Accessed: July 2012; Mirrored at `http://ontohub.org/repositories/tones`.

Information Systems Group (2012). Oxford ontology library. University of Oxford. Available at `http://www.cs.ox.ac.uk/isg/ontologies/`. Accessed: August 2012;.

Kazakov, Y. (2008). $\mathcal{RIQ}$ 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.

Kazakov, Y. (2009). Consequence-driven reasoning for Horn-$\mathcal{SHIQ}$ ontologies. In *Proc. 21st Int. Conf. on Artificial Intelligence (IJCAI'09)*, pp. 2040–2045. IJCAI.

Kazakov, Y. (2010). ConDOR project site. `https://code.google.com/p/condor-reasoner/`. Accessed: July 2013;.

Kazakov, Y., Krötzsch, M., & Simančík, F. (2012). Practical reasoning with nominals in the EL family of description logics. In *Proc. 13th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'12)*. AAAI Press.

Kazakov, Y., Krötzsch, M., & Simančík, F. (2014). The incredible ELK - from polynomial procedures to efficient reasoning with $\mathcal{EL}$ ontologies. *J. of Automated Reasoning*, *53*, 1–61.

Matentzoglu, N., Bail, S., & Parsia, B. (2013). A corpus of OWL DL ontologies. In *Proc. 26th Int. Workshop on Description Logics (DL'13)*, Vol. 1014. CEUR.

Matentzoglu, N., & Parsia, B. (2014). ORE 2014 reasoner competition dataset. Zenodo. Available at `http://dx.doi.org/10.5281/zenodo.10791`.

National Cancer Institute (2003). Nci thesaurus archive. Available at `http://ncit.nci.nih.gov/`. Accessed: December 2012;.

Ortiz, M., Rudolph, S., & Simkus, M. (2010). Worst-case optimal reasoning for the Horn-DL fragments of OWL 1 and 2. In *Proc. 12th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'10)*, pp. 269–279. AAAI Press.

Simančík, F. (2012). Elimination of complex RIAs without automata. In *Proc. 25th Int. Workshop on Description Logics (DL'12)*, Vol. 846. CEUR.

Simančík, F., Kazakov, Y., & Horrocks, I. (2011). Consequence-based reasoning beyond horn ontologies. In *Proc. 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI'11)*, pp. 1093–1098. IJCAI/AAAI.

Simančík, F., Motik, B., & Horrocks, I. (2014). Consequence-based and fixed-parameter tractable reasoning in description logics. *J. of Artificial Intelligence*, *209*, 29–77.

Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., & Katz, Y. (2007). Pellet: A practical OWL-DL reasoner. *J. of Web Semantics*, *5*(2), 51–53.

Smith, B., Ashburner, M., Rosse, C., Bard, J., Bug, W., Ceusters, W., Goldberg, L. J., Eilbeck, K., Ireland, A., Mungall, C. J., Consortium, T. O., Leontis, N., Rocca-Serra, P., Ruttenberg, A., Sansone, S.-A., Scheuermann, R. H., Shah, N., Whetzeland, P. L., & Lewis, S. (2007). The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. *J. of Nature Biotechnology*, *25*, 1251–1255.

Song, W., Spencer, B., & Du, W. (2012). WSReasoner: A prototype hybrid reasoner for ALCHOI ontology classification using a weakening and strengthening approach. In *Proc. 1st Int. Workshop on OWL Reasoner Evaluation (ORE'12)*, Vol. 858. CEUR.

Steigmiller, A., Glimm, B., & Liebig, T. (2013). Nominal schema absorption. In *Proc. 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI'13)*, pp. 1104–1110. AAAI Press.

Steigmiller, A., Glimm, B., & Liebig, T. (2014a). Coupling tableau algorithms for expressive description logics with completion-based saturation procedures. In *Proc. 7th Int. Joint Conf. on Automated Reasoning (IJCAR'14)*, Vol. 8562 of *LNCS*, pp. 449–463. Springer.

Steigmiller, A., Glimm, B., & Liebig, T. (2014b). Optimised absorption for expressive description logics. In *Proc. 27th Int. Workshop on Description Logics (DL'14)*, Vol. 1193. CEUR.

Steigmiller, A., Glimm, B., & Liebig, T. (2015). Completion graph caching for expressive description logics. In *Proc. 28th Int. Workshop on Description Logics (DL'15)*.

Steigmiller, A., Liebig, T., & Glimm, B. (2012). Extended caching, backjumping and merging for expressive description logics. In *Proc. 6th Int. Joint Conf. on Automated Reasoning (IJCAR'12)*, Vol. 7364 of *LNCS*, pp. 514–529. Springer.

Steigmiller, A., Liebig, T., & Glimm, B. (2014). Konclude: system description. *J. of Web Semantics*, *27*(1).

Tsarkov, D., & Horrocks, I. (2004). Efficient reasoning with range and domain constraints. In *Proc. 17th Int. Workshop on Description Logics (DL'04)*, Vol. 104. CEUR.

Tsarkov, D., & Horrocks, I. (2005). Optimised classification for taxonomic knowledge bases. In *Proc. 18th Int. Workshop on Description Logics (DL'05)*, Vol. 147. CEUR.

Tsarkov, D., & Horrocks, I. (2006). FaCT++ description logic reasoner: System description. In *Proc. 3rd Int. Joint Conf. on Automated Reasoning (IJCAR'06)*, Vol. 4130 of *LNCS*, pp. 292–297. Springer.

Tsarkov, D., Horrocks, I., & Patel-Schneider, P. F. (2007). Optimizing terminological reasoning for expressive description logics. *J. of Automated Reasoning*, *39*, 277–316.

W3C OWL Working Group (27 October 2009). *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation. Available at `http://www.w3.org/TR/owl2-overview/`.

Whetzel, P. L., Noy, N. F., Shah, N. H., Alexander, P. R., Nyulas, C., Tudorache, T., & Musen, M. A. (2011). BioPortal: enhanced functionality via new web services from the national center for biomedical ontology to access and use ontologies in software applications. *Nucleic Acids Research*, *39*(Web-Server-Issue), 541–545.

Zhou, Y., Cuenca Grau, B., Nenov, Y., Kaminski, M., & Horrocks, I. (2015). PAGOdA: Pay-as-you-go ontology query answering using a datalog reasoner. *J. of Artificial Intelligence Research*, *54*, 309–367.

Zhou, Y., Nenov, Y., Cuenca Grau, B., & Horrocks, I. (2014). Pay-as-you-go OWL query answering using a triple store. In *Proc. 28th AAAI Conf. on Artificial Intelligence (AAAI'14)*, pp. 1142–1148.