ulm university universität
uulm

Coupling Tableau Algorithms for the DL SROIQ with Completion-Based Saturation Procedures

Andreas Steigmiller, Birte Glimm, Thorsten Liebig

Ulmer Informatik-Berichte

Nr. 2014-02
Juli 2014

Coupling Tableau Algorithms for the DL $\mathcal{SROIQ}$ with Completion-Based Saturation Procedures

Technical Report

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

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

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

1 Introduction

The current version of the Web Ontology Language (OWL 2) [33] is based on the very expressive Description Logic (DL) $\mathcal{SROIQ}$ [9]. To handle (standard) reasoning tasks, sound and complete tableau algorithms are typically used, which are easily extensible and adaptable. Moreover, the use of a wide range of optimisation techniques allows for handling many expressive, real-world ontologies. Since standard reasoning tasks for $\mathcal{SROIQ}$ have N2EXPTIME-complete worst-case complexity [15], it is, however, not surprising that larger ontologies easily become unpractical for existing systems.

In contrast, the OWL 2 profiles define language fragments of $\mathcal{SROIQ}$ for which reasoning tasks can be realised efficiently, e.g., within polynomial worst-case complexity. For many of these language fragments specialised reasoning procedures have been developed, which are often based on a variant of saturation. For example, the OWL 2 EL profile is based on the DL $\mathcal{EL}^{++}$ which can be very efficiently handled by completion-based and consequence-based reasoning procedures [2,16]. These saturation algorithms have been further pushed to more expressive Description Logics (e.g., Horn-$\mathcal{SHIQ}$ [16]) for which they are often also able to outperform the more general tableau algorithms. In particular, they often allow 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), which makes these saturation-based procedures very fast. However, saturation algorithms have not yet been extended to very expressive DLs such as $\mathcal{SROIQ}$, mainly due to difficulties caused by the complete handling of cardinality restrictions.

Unfortunately, due to their different nature, it is not easily possible to combine tableau algorithms and saturation procedures. 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 or they have to face possible performance losses by using more general reasoning systems. This is especially unfavourable if such language features are only required for few axioms in the ontologies, because then the completeness is not further ensured for the specialised procedures and the fully-fledged reasoners, which are typically based on tableau algorithms, are possibly not efficient enough to handle such ontologies in practice. Obviously, for such cases, reasoning systems with better pay-as-you-go behaviours are required, where the part of the ontology that is not affected by the axioms outside the traceable fragment can still be handled efficiently.

Recently, new approaches have been proposed to improve the reasoning performance for ontologies of more expressive Description Logics by combining saturation procedures and fully-fledged reasoners in a black box manner [1,25]. These approaches try to delegate as much work as possible to the specialised and more efficient reasoner, which allows for reducing the workload of the fully-fledged tableau algorithm, and often results in a better pay-as-you-go behaviour than using a tableau reasoner alone.

In this paper, we present a much tighter coupling between saturation and tableau algorithms, whereby further performance improvements are achieved. After introducing some preliminaries (Section 2), we present a saturation procedure that is adapted to the data structures of a tableau algorithm (Section 3). This allows for easily passing information between the saturation and the tableau algorithm within the same reasoning system. Moreover, the saturation partially handles features of more expressive Description Logics 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).

2 Preliminaries

Since our approach aims at assisting fully-fledged reasoners which are usually based on tableau calculi for $\mathcal{SROIQ}$, we first give a brief introduction into the DL $\mathcal{SROIQ}$ (see [3] for a detailed introduction into DLs), and then we describe a tableau algorithm as it is typically used by reasoning systems (see [9] for details).

2.1 The Description Logic $\mathcal{SROIQ}$

We first define the syntax of roles, concepts, and individuals, and then we go on to axioms and ontologies/knowledge bases. Additionally, we define typically used restrictions for the combination of the different axioms, which are necessary to ensure the

decidability for many inference problems of $\mathcal{SROIQ}$. Subsequently, we define the semantics of these components.

Definition 1 (Syntax of $\mathcal{SROIQ}$). *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 $a \in N_I, n \in \mathbf{N}_0, A \in N_C$, and $r \in \mathsf{Rols}(\Sigma)$:*

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

We use roles, concepts and individuals to build axioms of ontologies 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* general role inclusion *(GRI) 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* general role assertion *(GRA) axiom is of the form* $\mathsf{Disj}(r, s)$ *or* $\mathsf{Refl}(r)$*. An RBox is a finite set of GRIs and GRAs. 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 *$\mathcal{K}$ 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 kinds of GRAs in the RBox, e.g., axioms that specify roles as transitive, symmetric, or irreflexive. However, such axioms can be indirectly expressed in other ways and, therefore, we omit their presentation here. 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 will also utilise below to eliminate all ABox assertions. 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., this role is implied by all other roles. In the following, we will 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 $\mathcal{T} \in \mathcal{K}$.

As mentioned, if we arbitrarily combine the axioms of Definition 2, then we easily run into decidability issues. In order to ensure termination for standard reasoning tasks such as satisfiability testing for concepts, we have to restrict the role inclusion axioms to be regular and, in addition, we allow some concept expressions only in combination with simple roles, which is described below in more detail.

Definition 3 (Regularity of Role Inclusion Axioms). *A set of GRIs is* regular *if the used role names can be sorted in a strict partial order and all GRIs are regular. Let $\prec$ be a strict partial order on role names, then for the role names $r, s_1, \ldots, s_n$, the GRI axiom $u \sqsubseteq r$ is regular if*

1. $u = r \circ r$, *or*
2. $u = r^-$, *or*
3. $u = s_1 \circ \ldots \circ s_n$ *and* $s_i \prec r$ *for all* $1 \leq i \leq n$, *or*
4. $u = r \circ s_1 \circ \ldots \circ s_n$ *and* $s_i \prec r$ *for all* $1 \leq i \leq n$, *or*
5. $u = s_1 \circ \ldots \circ s_n \circ r$ *and* $s_i \prec r$ *for all* $1 \leq i \leq n$.

Now, we can define simple and complex roles and, in order to ensure decidability, we only allow simple roles in concepts of the form $\geqslant n\, r.C$, $\leqslant n\, r.C$, and $\exists r.\mathsf{Self}$. In addition, we require that all $\mathsf{Disj}(r, s)$ axioms are only using simple roles.

Definition 4 (Simple and Complex Roles). *For a set of GRI axioms, we call a role* simple *if it is not complex. A role r is called* complex *w.r.t. a set of GRIs if*

1. *its inverse role is complex, or*
2. *it occurs on the right-hand side of a GRI axiom of the form $s_1 \circ \ldots \circ s_n \sqsubseteq r$ and s_i is complex for $1 \leq i \leq n$ or $n > 1$.*

In the remainder of the paper, we assume that all knowledge bases comply the presented restrictions on regularity and simple roles.

Given a set of role inclusion axioms (e.g., in form of an RBox), we use $\sqsubseteq^*$ as the transitive-reflexive closure over all $r \sqsubseteq s$ and $\mathsf{inv}(r) \sqsubseteq \mathsf{inv}(s)$ axioms in the RBox. We call a role r a sub-role of s and s a super-role of r if $r \sqsubseteq^* s$.

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

Definition 5 (Semantics of $\mathcal{SROIQ}$-concepts). *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.

Definition 6 (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 a GRI $s_1 \circ \ldots \circ s_n \sqsubseteq r$ 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 a GRA of the form $\mathsf{Disj}(r, s)$ and all $r^{\mathcal{I}} \cap s^{\mathcal{I}} = \emptyset$, 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 GRI/GRA axiom in $\mathcal{R}$, each assertion in $\mathcal{A}$). We say that $\mathcal{I}$* satisfies *$\mathcal{K}$ 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

For ease of presentation, we assume in the remainder of the paper that all concepts are in negation normal form (NNF). Each concept can be transformed into an equivalent one in NNF by pushing negation inwards, making use of de Morgan's laws and the following equivalences that exploit the duality between existential and universal restrictions, and between at-most and at-least cardinality restrictions [12]:

$$\begin{aligned} \neg(\forall r.C) &\equiv \exists r.\neg C & \neg(\exists r.C) &\equiv \forall r.\neg C \\ \neg(\leqslant n\, r.C) &\equiv \geqslant (n+1)\, r.C & \neg(\geqslant 0\, r.C) &\equiv \bot \\ \neg(\geqslant k\, r.C) &\equiv \leqslant (k-1)\, r.C, \end{aligned}$$

where $k \in \mathbf{N}$ and $n \in \mathbf{N}_0$. For C a concept possibly not in NNF, let $\mathsf{nnf}(C)$ be the equivalent concept to C in NNF.

In the following, we also assume that all ABox axioms are "internalised" into the TBox of a knowledge base, which can be easily realised in the presence of nominals, e.g., by expressing a concept assertion $C(a)$ (role assertion $r(a, b)$) as $\{a\} \sqsubseteq C$ ($\{a\} \sqsubseteq \exists r.\{b\}$). Moreover, since the tableau algorithm only supports TBox axioms of the form $(A_1 \sqcap A_2) \sqsubseteq C$ and $H \sqsubseteq C$ with $H = A$, $H = \{a\}$, or $H = \top$, all GCIs that do not match these forms have to be "internalised". A not supported GCI $C \sqsubseteq D \in \mathcal{T}$ can be internalised by adding the axiom $\top \sqsubseteq \mathsf{nnf}(\neg C \sqcup D)$ to $\mathcal{T}$, which can then be handled by the tableau algorithm. Obviously, such an internalisation creates (possibly many) disjunctions of the form $C \sqcup D$, which causes non-determinism in the tableau algorithm and easily decreases the reasoning performance. To counteract this, a preprocessing step called absorption is often used (see Section 2.4), which significantly reduces the number of concepts that has to be internalised.

Moreover, we assume that all universal restrictions that occur in a knowledge base $\mathcal{K}$ are normalised such that complex role inclusion axiom of the form $s_1 \circ \ldots \circ s_n \sqsubseteq r$ with $n > 1$ are implicitly handled by additional axioms. Hence, we do not require further adjustments in the tableau algorithm to handle propagations over complex roles.

Definition 7 (Normalisation of Universal Restrictions). *Let $\mathcal{K}$ be a knowledge base that contains the set of GRIs R. For $A, B, F_1, \ldots, F_n, F'_1, \ldots, F'_n$ atomic concepts, we say*

a knowledge base $\mathcal{K}$ contains all propagation axioms *for an axiom of the form* $A \sqsubseteq \forall r.B$ *w.r.t. the role name* r *if for every GRI* $u \sqsubseteq r \in R$ *the following conditions holds:*

1. *if* $u = r \circ r$, *then* $B \sqsubseteq A \in \mathcal{K}$;
2. *if* $u = r^-$, *then* $\mathcal{K}$ *contains all propagation axioms for* $A \sqsubseteq \forall r^-.B$;
3. *if* $u = s_1 \circ \ldots \circ s_n$, *then* $\mathcal{K}$ *contains axioms of the form* $A \sqsubseteq F_1$, $F'_1 \sqsubseteq F_2, \ldots, F'_{n-1} \sqsubseteq F_n$, $F'_n \sqsubseteq B$, *and, for all* $1 \leq i \leq n$, *the axiom* $F_i \sqsubseteq \forall s_i.F'_i$ *for which also all propagation axioms are contained by* $\mathcal{K}$;
4. *if* $u = r \circ s_1 \circ \ldots \circ s_n$, *then* $\mathcal{K}$ *contains axioms of the form* $B \sqsubseteq F_1$, $F'_1 \sqsubseteq F_2, \ldots, F'_{n-1} \sqsubseteq F_n$, $F'_n \sqsubseteq B$, *and, for all* $1 \leq i \leq n$, *the axiom* $F_i \sqsubseteq \forall s_i.F'_i$ *for which also all propagation axioms are contained by* $\mathcal{K}$;
5. *if* $u = s_1 \circ \ldots \circ s_n \circ r$, *then* $\mathcal{K}$ *contains axioms of the form* $A \sqsubseteq F_1$, $F'_1 \sqsubseteq F_2, \ldots, F'_{n-1} \sqsubseteq F_n$, $F'_n \sqsubseteq A$, *and, for all* $1 \leq i \leq n$, *the axiom* $F_i \sqsubseteq \forall s_i.F'_i$ *for which also all propagation axioms are contained by* $\mathcal{K}$.

Analogously for inverse roles, we say that $\mathcal{K}$ contains all propagation axioms *for an axiom of the form* $A \sqsubseteq \forall r^-.B$ *w.r.t. the role name* r *if for every GRI* $u \sqsubseteq r \in R$ *the following conditions holds:*

1. *if* $u = r \circ r$, *then* $B \sqsubseteq A \in \mathcal{K}$;
2. *if* $u = r^-$, *then* $\mathcal{K}$ *contains all propagation axioms for* $A \sqsubseteq \forall r.B$;
3. *if* $u = s_1 \circ \ldots \circ s_n$, *then* $\mathcal{K}$ *contains axioms of the form* $A \sqsubseteq F_n$, $F'_n \sqsubseteq F_{n-1}, \ldots, F'_2 \sqsubseteq F_1$, $F'_1 \sqsubseteq B$, *and, for all* $1 \leq i \leq n$, *the axiom* $F_i \sqsubseteq \forall s_i.F'_i$ *for which also all propagation axioms are contained by* $\mathcal{K}$;
4. *if* $u = r \circ s_1 \circ \ldots \circ s_n$, *then* $\mathcal{K}$ *contains axioms of the form* $A \sqsubseteq F_n$, $F'_n \sqsubseteq F_{n-1}, \ldots, F'_2 \sqsubseteq F_1$, $F'_1 \sqsubseteq A$, *and, for all* $1 \leq i \leq n$, *the axiom* $F_i \sqsubseteq \forall s_i.F'_i$ *for which also all propagation axioms are contained by* $\mathcal{K}$;
5. *if* $u = s_1 \circ \ldots \circ s_n \circ r$, *then* $\mathcal{K}$ *contains axioms of the form* $B \sqsubseteq F_n$, $F'_n \sqsubseteq F_{n-1}, \ldots, F'_2 \sqsubseteq F_1$, $F'_1 \sqsubseteq B$, *and, for all* $1 \leq i \leq n$, *the axiom* $F_i \sqsubseteq \forall s_i.F'_i$ *for which also all propagation axioms are contained by* $\mathcal{K}$.

For a possibly inverse role r *and a concept* C, *let* $\forall r.C$ ($\exists r.\neg C$) *be a universal (existential) restriction that occurs in a knowledge base* $\mathcal{K}$. *We say that* $\forall r.C$ ($\exists r.\neg C$) *is* normalised *w.r.t.* $\mathcal{K}$ *if*

- C *is an atomic concept,* $\forall r.C$ ($\exists r.\neg C$) *occurs only in axioms of the form* $A \sqsubseteq \forall r.C$, *and* $\mathcal{K}$ *contains all propagation axioms for* $A \sqsubseteq \forall r.C$, *or*
- $\mathcal{K}$ *contains axioms of the form* $\forall r.C \sqsubseteq A$, $A \sqsubseteq \forall r.B$, $B \sqsubseteq C$, *where* A, B *are atomic concepts, and* $\mathcal{K}$ *contains all propagation axioms for* $A \sqsubseteq \forall r.B$.

Obviously, the normalisation of a knowledge base $\mathcal{K}$ with respect to the universal restrictions that occur in $\mathcal{K}$ (possibly in negated form) can be generated with a recursive function that introduces the propagation axioms with fresh atomic concepts over complex roles as defined above (under the assumption that the role inclusion axioms of $\mathcal{K}$ are regular). Note, the normalisation of universal restrictions might exponentially blow up the knowledge base. Although such a blow up cannot be avoided in the worst-case [15], it is usually not a problem for real-world ontologies. Nevertheless, many reasoning systems create the required concepts and axioms only on demand, i.e., only if these

concepts are used by the tableau algorithm, which is for example possible by using an automata approach [9]. In practice, the normalisation of universal restrictions is often further optimised. For example, it is obviously not necessary to normalise universal restrictions for simple roles.

In the remainder of the paper, we assume that all knowledge bases are normalised, i.e., all concepts occurring in the knowledge base are in NNF, all universal restrictions in these concepts are normalised and not supported axioms are internalised. For a concept C, which is possibly not normalised, we use $\mathsf{norm}(C)$ to get the normalised concept of C. Note, the normalisation of universal restrictions possibly introduces new axioms, but the knowledge base can be easily fixed by creating this normalisation for all concept that possibly occur in the tableau algorithm in a preprocessing step.

2.3 Tableau Algorithm for $\mathcal{SROIQ}$

Model construction calculi, such as tableau, 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".

Definition 8 (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., $\mathsf{cons}_\mathcal{K} = \{C, D \mid C \sqsubseteq D \in \mathcal{T}\}$. We define the* closure $\mathsf{clos}(\mathcal{K})$ of $\mathcal{K}$ *as:*

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

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 9 (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, $\mathsf{mneighbs}(v, r, C)$ as $\{v' \in V \mid v'$ is an r-neighbour of v and $C \in \mathcal{L}(v')\}$.

To test the consistency of a knowledge base, the completion graph is initialised for the tableau algorithm by creating one node for each individual/nominal in the input knowledge base. 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 $G = (\{v_1, \ldots, v_\ell\}, E, \mathcal{L}, \emptyset)$ 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 $1 \leq i \leq \ell$.

Note, many inference problems for the DL $\mathcal{SROIQ}$ can be easily reduced to consistency checking. 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 tableau algorithm works by decomposing 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 10 (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. $\{C, \mathsf{norm}(\neg C)\} \subseteq \mathcal{L}(v)$ *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* [10] 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 11 (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 $\mathcal{L}(v') = \mathcal{L}(w')$,*

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

$\sqsubseteq_1$-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	$\exists r.\mathsf{Self} \in \mathcal{L}(v)$ or $\mathsf{Refl}(r) \in \mathcal{K}$, 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 $\{C, \mathsf{norm}(\neg C)\} \cap \mathcal{L}(v') = \emptyset$
	then	$\mathcal{L}(v') = \mathcal{L}(v') \cup \{H\}$ for some $H \in \{C, \mathsf{norm}(\neg C)\}$
$\geqslant$-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 $1 \leq i < j \leq n$
	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 $1 \leq i < j \leq m$.

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.

During the expansion it is sometimes necessary to merge two nodes or to delete (prune) a part of the completion graph. When a node w is merged into a node v (e.g., by an application of the $\leqslant$-rule), we "prune" the completion graph by removing w and, recursively, all blockable successors of w to prevent a further rule application on these nodes.

Intuitively, when we merge a node w into a node 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.

Definition 12 (Pruning, Merging). *Pruning a node w in the completion graph $G = (V, E, \mathcal{L}, \dot{\neq})$, written* $\mathsf{prune}(w)$*, yields a graph that is obtained from G as follows:*

1. *for all successors v' of w, remove $\langle w, v'\rangle$ from E and, if v' is blockable,* $\mathsf{prune}(v')$*;*
2. *remove w from V.*

Merging a node w into a node v in $G = (V, E, \mathcal{L}, \dot{\neq})$, written $\mathsf{merge}(w, v)$*, yields a graph that is obtained from G as follows:*

1. *for all nodes v' such that $\langle v', w\rangle \in E$*
   (a) *if $\{\langle v, v'\rangle, \langle v', v\rangle\} \cap E = \emptyset$, then add $\langle v', v\rangle$ to E and set $\mathcal{L}(\langle v', v\rangle) = \mathcal{L}(\langle v', w\rangle)$,*
   (b) *if $\langle v', v\rangle \in E$, then set $\mathcal{L}(\langle v', v\rangle) = \mathcal{L}(\langle v', v\rangle) \cup \mathcal{L}(\langle v', w\rangle)$,*
   (c) *if $\langle v, v'\rangle \in E$, then set $\mathcal{L}(\langle v, v'\rangle) = \mathcal{L}(\langle v, v'\rangle) \cup \{\mathsf{inv}(r) \mid r \in \mathcal{L}(\langle v', w\rangle)\}$, and*
   (d) *remove $\langle v', w\rangle$ from E;*
2. *for all nominal nodes v' such that $\langle w, v'\rangle \in E$*
   (a) *if $\{\langle v, v'\rangle, \langle v', v\rangle\} \cap E = \emptyset$, then add $\langle v, v'\rangle$ to E and set $\mathcal{L}(\langle v, v'\rangle) = \mathcal{L}(\langle w, v'\rangle)$,*
   (b) *if $\langle v, v'\rangle \in E$, then set $\mathcal{L}(\langle v, v'\rangle) = \mathcal{L}(\langle v, v'\rangle) \cup \mathcal{L}(\langle w, v'\rangle)$,*
   (c) *if $\langle v', v\rangle \in E$, then set $\mathcal{L}(\langle v', v\rangle) = \mathcal{L}(\langle v', v\rangle) \cup \{\mathsf{inv}(r) \mid r \in \mathcal{L}(\langle w, v'\rangle)\}$, and*
   (d) *remove $\langle w, v'\rangle$ from E;*
3. *$\mathcal{L}(v) = \mathcal{L}(v) \cup \mathcal{L}(w)$;*
4. *add $v \dot{\neq} v'$ for all v' such that $w \dot{\neq} v'$; and*
5. $\mathsf{prune}(w)$.

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. In addition, it is necessary to associate a level with those nominal nodes that are newly created by the NN-rule and to apply the crucial rules first to nominal nodes with lower levels. Basically, we define the level of a nominal node as the length of the shortest path to a node that contains a nominal for an original individual in its label.

Definition 13 (Level of Nominal Nodes). *Let $a_1, \ldots, a_n$ be all individuals of a knowledge base $\mathcal{K}$. The level of a nominal node v in a completion graph G for $\mathcal{K}$ is defined as*

- 0 *if* $\{a_i\} \in \mathcal{L}(v)$ *with* $1 \leq i \leq n$, *or*
- $i+1$ *if v has a neighbour node* v' *that has the level i and there is no neighbour node with a level below i.*

2.4 (Binary) Absorption

Absorption is used as a preprocessing step in order to reduce the non-determinism in the tableau algorithm. Basically, axioms are rewritten in possibly several simpler concept inclusion axioms for which the lazy unfolding rules $\sqsubseteq_1$ and $\sqsubseteq_2$ can be used in the tableau algorithm and, therefore, internalisation is not required. Algorithms based on binary absorption [14] allow and create axioms of the form $(A_1 \sqcap A_2) \sqsubseteq C$, whereby also more complex axioms can be absorbed. A binary absorption axiom $(A_1 \sqcap A_2) \sqsubseteq C$ can be efficiently handled by adding C only to node labels if A_1 and A_2 are already present, which is realised by the $\sqsubseteq_2$-rule of our tableau algorithm. More sophisticated absorption algorithms, such as partial absorption [26,27], are further improving the handling of knowledge bases for more expressive Description Logics since the non-determinism that is caused by disjunctions on the right-hand side of axioms is further reduced. Roughly speaking, the non-absorbable disjuncts are partially used as conditions on the left-hand side of additional inclusion axioms such that the processing of the disjunctions can further be delayed.

Many state-of-the-art reasoning systems are at least using some kind of binary absorption, which makes the processing of simple ontologies (e.g., EL ontologies) also with the tableau algorithm deterministic. In the following, we assume that knowledge bases are, at least, preprocessed with a variant of binary absorption and we also use the syntax of binary absorption axioms to illustrate the algorithms and examples. However, for our optimisations, a detailed understanding of a (binary) absorption algorithm is not necessary and, therefore, we only present its principle in the following example.

Example 1. For the TBox $\mathcal{T}_1 = \{A_1 \sqsubseteq A_2 \sqcap \exists r.A_3, A_3 \sqsubseteq A_1, A_2 \sqcap \exists r.A_1 \sqsubseteq B\}$, all of the axioms except the GCI $A_2 \sqcap \exists r.A_1 \sqsubseteq B$ are of the form $A \sqsubseteq C$ and, therefore, they can be efficiently handled in the tableau algorithm by the $\sqsubseteq_1$-rule. In contrast, the GCI $A_2 \sqcap \exists r.A_1 \sqsubseteq B$ would, without absorption, be handled as $\top \sqsubseteq \neg A_2 \sqcup \forall r.\neg A_1 \sqcup B$ and, as a consequence, the tableau algorithm would have to process the obtained disjunction for every node in the completion graph. The absorption rewrites $A_2 \sqcap \exists r.A_1 \sqsubseteq B$ into the axioms $A_1 \sqsubseteq \forall r^-.F_1$ and $(A_2 \sqcap F_1) \sqsubseteq B$, where F_1 is a fresh atomic concept that is used to preserve the semantics of the original axiom. In principle, the absorption recursively generates simple concept inclusion axioms that imply a fresh atomic concept if a (sub-)concept of the left-hand side of an axiom is satisfied. For example, F_1 is implied if $\exists r.A_1$ is satisfied. This is continued until the complete left-hand side is absorbed and the right-hand side of the axiom can be implied by the last axiom generated with the absorption algorithm. Hence, from the absorption of $\mathcal{T}_1$, we obtain a new TBox $\mathcal{T}_1'$ consisting of the axioms $A_1 \sqsubseteq A_2 \sqcap \exists r.A_3$, $A_3 \sqsubseteq A_1$, $A_1 \sqsubseteq \forall r^-.F_1$ and $(A_2 \sqcap F_1) \sqsubseteq B$, which can now be deterministically handled in the tableau algorithm by lazy unfolding rules.

3 Saturation Compatible with Tableau Algorithms

In this section, we describe a saturation method that is an adaptation of the completion-based procedure [2] such that it generates data structures that are compatible for further usage within a fully-fledged tableau algorithm for more expressive Description Logics. 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 the completion graph, labelled with sets of concepts. The saturated labels can then be used to initialise the labels of new nodes in the completion graph or to block the processing of successors. Moreover, in some cases, it is directly possible to extract completion graphs from the data structures of the saturation, which makes the model construction with the tableau algorithm unnecessary.

Note, the adapted saturation method is not designed to cover a certain OWL 2 profile or a specific Description Logic language. In contrast, we saturate those parts of knowledge bases that are easily supportable with an efficient algorithm (see Section 3.1), i.e., we simply ignore unsupported concept constructors, 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.

As mentioned, the saturation is intended to generate compatible node labels to avoid conversions and, therefore, it has to operate on the same representation of the knowledge base, i.e., the saturation as well as the tableau algorithm have to apply rules on concepts and axioms that are obtained with the same normalisation and preprocessing approach. In principle, the saturation as well as the tableau algorithm can be adapted to the commonly used representation of the other approach. However, since tableau algorithms are usually equipped with a wide range of optimisations, we prefer a saturation method that is compatible to the commonly used knowledge base representation for tableau calculi. This allows for integrating the saturation optimisations into the often more complex reasoning systems for expressive Description Logics based on tableau algorithms with only minor modifications.

As hitherto, we assume for the presentation of the saturation procedure that the knowledge bases are preprocessed as shown in Section 2 for the introduced tableau algorithm. In particular, this means that all concepts are in NNF and all universal restrictions are normalised such that complex roles are considered without the introduction of adjusted rules in the reasoning procedures. Although absorption is not a prerequisite for the saturation procedure (and in principle also not for the tableau algorithm), we assume that all GCIs are absorbed as good as possible. Since the absorption rewrites axioms such that the non-determinism is reduced and the saturation only handles deterministic parts of the knowledge base, a good absorption algorithm significantly improves the presented approach. To be more precise, in the following, we assume that the knowledge bases are at least absorbed with binary absorption (see Section 2.4), which allows for a deterministic handling of EL ontologies.

Table 2. Basic saturation rules

$\sqsubseteq_1$-rule:	if $H \in \mathcal{L}(v)$, $H \sqsubseteq C \in \mathcal{K}$ with $H = A$, or $H = \{a\}$, or $H = \top$, and $C \notin \mathcal{L}(v)$, then $\mathcal{L}(v) \longrightarrow \mathcal{L}(v) \cup \{C\}$
$\sqsubseteq_2$-rule:	if $\{A_1, A_2\} \subseteq \mathcal{L}(v)$, $(A_1 \sqcap A_2) \sqsubseteq C \in \mathcal{K}$, and $C \notin \mathcal{L}(v)$, then $\mathcal{L}(v) \longrightarrow \mathcal{L}(v) \cup \{C\}$
$\sqcap$-rule:	if $C_1 \sqcap C_2 \in \mathcal{L}(v)$ and $\{C_1, C_2\} \not\subseteq \mathcal{L}(v)$, then $\mathcal{L}(v) \longrightarrow \mathcal{L}(v) \cup \{C_1, C_2\}$
$\exists$-rule:	if $\exists r.C \in \mathcal{L}(v)$ and $r \notin \mathcal{L}(\langle v, v_C \rangle)$, then $\mathcal{L}(\langle v, v_C \rangle) \longrightarrow \mathcal{L}(\langle v, v_C \rangle) \cup \{r\}$
$\forall$-rule:	if $\forall r.C \in \mathcal{L}(v)$, there is an $\mathsf{inv}(r)$-predecessor v' of v , and $C \notin \mathcal{L}(v')$, then $\mathcal{L}(v') \longrightarrow \mathcal{L}(v') \cup \{C\}$

3.1 Saturation based on Tableau Rules

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

Definition 14 (Saturation Graph). *A* saturation graph *for $\mathcal{K}$ is a directed graph $S = (V, E, \mathcal{L})$ with the nodes $V \subseteq \{v_C \mid C \in \mathsf{fclos}(\mathcal{K})\}$. Each node $v_C \in V$ is labelled with a set $\mathcal{L}(v) \subseteq \mathsf{fclos}(\mathcal{K})$ 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 $\doteq$ relation, which can be omitted since the saturation is not designed to completely handle cardinality restrictions and, therefore, we also do not need to keep track of inequalities between nodes in the saturation graph. Furthermore, each node in the saturation graph is the representative node for a specific concept, which allows for "reusing" nodes 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.

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

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

For the initialised saturation graph, we apply the saturation rules depicted in Table 2. Note that if a saturation rule refers to the representative node for a concept C and the node v_C does not yet exist, then we assume that the saturation graph is automatically extended by this node. Although the saturation rules are very similar to the

Table 3. Extended saturation rules

$\sqcup$-rule:	if	$C_1 \sqcup C_2 \in \mathcal{L}(v)$, there is some $D \in \mathcal{L}(v_{C_1}) \cap \mathcal{L}(v_{C_2})$, and $D \notin \mathcal{L}(v)$,
	then	$\mathcal{L}(v) \longrightarrow \mathcal{L}(v) \cup \{D\}$
$\geqslant$rule:	if	$\geqslant n\, r.C \in \mathcal{L}(v)$ with $n \geq 1$ and $r \notin \mathcal{L}(\langle v, v_C \rangle)$,
	then	$\mathcal{L}(\langle v, v_C \rangle) \longrightarrow \mathcal{L}(\langle v, v_C \rangle) \cup \{r\}$
Self-rule:	if	$\exists r.\mathsf{Self} \in \mathcal{L}(v)$ or $\mathsf{Refl}(r) \in \mathcal{K}$ and v is not an r-successor of v,
	then	$\mathcal{L}(v) \longrightarrow \mathcal{L}(\langle v, v \rangle) \cup \{r\}$
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	$\mathcal{L}(v) \longrightarrow \mathcal{L}(v) \cup \{D\}$
$\bot$-rule:	if	$\bot \notin \mathcal{L}(v)$, and
		1. $\{C, \neg C\} \subseteq \mathcal{L}(v)$, or
		2. v is an r-successor of v and $\neg\exists r.\mathsf{Self} \in \mathcal{L}(v)$, 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 and $r \sqsubseteq^{*} s$ 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 exist a successor node v' of v with $\bot \in \mathcal{L}(v')$, or
		9. there exist a node $v_{\{a\}}$ with $\bot \in \mathcal{L}(v_{\{a\}})$,
	then	$\mathcal{L}(v) \longrightarrow \mathcal{L}(v) \cup \{\bot\}$

corresponding expansion rules in the tableau algorithm, there are some differences. For example, the number of nodes is limited by the number of (sub-)concepts occurring in the knowledge base due to the reuse of nodes for satisfying existentially restricted concepts. Consequently, the saturation is terminating since the rules are only applied when they can add new concepts or roles to node or edge labels. Moreover, a cycle detection such as blocking is not required, which makes the rule application very fast. Note also that the $\forall$-rule propagates concepts only to the predecessors of a node, which is necessary in order to allow the reuse of nodes for existentially restricted concepts. If the concepts were also propagated into the successor direction, then we would have to create and saturate new nodes that differ by the propagated concepts. Furthermore, in the presence of cyclic concepts, it would be necessary to use an appropriate blocking condition such that the creation of new nodes can be stopped. Hence, due to the omitted propagation to successors, we obtain a faster saturation procedure, but the supported expressiveness slips to a subset of the DL Horn-$\mathcal{ALCHI}$. To be more precise, if existential restrictions with inverse roles or universal restrictions are used in the knowledge base such that they can propagate concepts to successors for any node in the saturation graph, then our procedure is possibly incomplete. However, this is still sufficient for the DL $\mathcal{EL}$ and several of its extensions. In principle, the reuse of nodes is also possible in algorithms for more expressive Description Logics, but then only in a non-deterministic way [20].

To improve the saturation support for more expressive Description Logics, e.g,. the DL $\mathcal{SROIQ}$, the saturation rules can be extended by the rules of Table 3. Again, the

primary objective of our saturation is to efficiently derive as many sound inferences as possible and afterwards we check where the saturation graph is possibly incomplete. Although there are often several ways to integrate the support of more expressive concept constructors, we chose a simple one that only allows a partial saturation, but still can be implemented very efficiently. For instance, the $\sqcup$-rule adds only those concepts that are implied by both disjuncts. Obviously, the addition of concepts by this rule is sound, but the handling of disjunctions of the form $C_1 \sqcup C_2$ is often incomplete, since the $\sqcup$-rule application does usually not directly add a disjunct and the disjuncts are possibly also not added in other ways. Similarly to the $\exists$-rule, we build the edges to (possibly reused) successor nodes for at-least cardinality restrictions. Thereby, the cardinality is ignored by the $\geqslant$-rule, 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 those concepts that are derived for $v_{\{a\}}$ or for descendant nodes that also have $\{a\}$ in their label (instead of merging such nodes as in tableau procedures). 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 with the saturation. However, the implementation is very simple and does not require the repeated saturation of the same concepts extended by small influences from the nominals. Of course, there are extensions possible that still enable a polynomial saturation algorithm with nominals, e.g., reasoning procedures for the DL $\mathcal{EL}^{++}$ [17], but, as of now, many less expressive ontologies are using the nominals in such a simple way that this o-rule is already sufficient (e.g., by using nominals only in concepts of the form $\exists r.\{a\}$), and for many more expressive ontologies, the saturation is also in other ways incomplete. The support for $\exists r.\mathsf{Self}$ concepts is straightforward by simply adding a Self-rule that makes the corresponding node to an r-successor of itself.

In contrast to tableau algorithms, we also need a $\bot$-rule. This is typical for saturation procedures since we are interested in associating clashes with specific nodes instead of entire completion graphs. As a consequence, the saturation allows for handling several independent concepts in a one-pass manner within the same saturation graph and to, nevertheless, distinguish unsatisfiable nodes from nodes that are (possibly) still satisfiable. The $\bot$-rule adds the concept $\bot$ to the label of those nodes for which a clash can be discovered. Furthermore, it propagates $\bot$ to the ancestor nodes and, in case $\bot$ is in its label of a node that represents an nominal, then the knowledge base is inconsistent and $\bot$ is also propagated to every other node label in the saturation graph.

The rules also include a $\bot$-rule, which adds the concept $\bot$ to the label of those nodes for which a clash can be discovered. Furthermore, it propagates $\bot$ to the ancestor nodes. In case $\bot$ occurs in the label of a representative node for a nominal, the knowledge base is inconsistent and $\bot$ is propagated to every node label in the saturation graph; otherwise $\bot$ in the label of a node v_C indicates the unsatisfiability of C. The conditions that trigger the $\bot$-rule application are very similar to the clash conditions for completion graphs for the DL $\mathcal{SROIQ}$ (cf. Definition 10). However, since the saturation graph does not contain an $\dot{\neq}$ relation, we do not have conditions for clashes related to differently stated individuals by the $\dot{\neq}$ relation (e.g., for combinations with nominals). In addition, the detection of clashes for combinations with $\mathsf{Disj}(r, s)$ axioms is more difficult due to the reuse of nodes. For example, a node v_C can be an r- and an s-successor if the concepts

$\exists r.C$ and $\exists s.C$ are in the label of the same node. Therefore, we have some modified clash conditions for combinations with $\mathsf{Disj}(r, s)$ axioms. Note, the saturation does not merge nodes with the same nominals in their label. However, since the merging of such nodes is in principle required, we are, at least, interested in the detection of some obvious clashes that would be discovered for the merging of these nodes. In particular, the merging of nominal nodes also merges the labels of the edges from and to these nodes and, for the DL $\mathcal{SROIQ}$, this can cause clashes due to concepts of the form $\neg\exists r.\mathsf{Self}$ and $\mathsf{Disj}(r, s)$ axioms. Basically, if a node v has an r-successor v' and both nodes have the same nominal in their label, then the merging of these nodes would replace the r-edge between the nodes with an r-loop, i.e., v would be an r-successor of itself. If concepts of the form $\neg\exists r.\mathsf{Self}$ or $\neg\exists r^{-}.\mathsf{Self}$ are in the label of v, then the merging of v and v' would cause a clash, which is considered with Condition 3 in the $\bot$-rule. In addition, clashes caused by the merging of an r-successor node v with an s-successor node v', where $\{a\} \in \mathcal{L}(v) \cap \mathcal{L}(v')$, are considered by Condition 7 of the $\bot$-rule. Although 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 propagated to a successor node v, where v has already the negation of C in its the label), but the presented conditions of the $\bot$-rule are already sufficient to show the completeness. Hence, we omitted further clash conditions for ease of presentation.

Example 2. Let us assume that the TBox $\mathcal{T}_2$ contains the following axioms:

$$A_1 \sqsubseteq \exists s^{-}.A_2 \qquad A_1 \sqsubseteq A_2 \sqcup \{a\} \qquad A_2 \sqsubseteq B \qquad A_2 \sqsubseteq \exists s.\{a\}$$
$$A_2 \sqsubseteq \leqslant 1\, s.B \qquad \{a\} \sqsubseteq B \qquad \{a\} \sqsubseteq \geqslant 2\, r.A_2$$

For a saturation graph that is initialised with the representative node for the concept A_1, the application of the rules of Table 2 and 3 generates a saturation graph as depicted in Figure 1. Note, the saturation procedure starts the rule application on node v_{A_1} and creates other nodes on demand, e.g., for the processing of nominals, disjunctions, existential restrictions, and at-least cardinality restrictions. Although the concept B is added to node labels, a node for B is not created since B is not used in a way that requires this.. Also note that the $\sqcup$-rule application adds the concept B to the label of v_{A_1}, because B is in the label of the representative nodes for both disjuncts of the disjunction $A_2 \sqcup \{a\}$ (i.e., $B \in \mathcal{L}(v_{A_2}) \cap \mathcal{L}(v_{\{a\}})$).

With a suitable absorption technique, the saturation is usually able to derive and add the majority of those concepts that would also be added by the tableau algorithm for an equivalent node. This is especially the case for ontologies that primarily use features of the DL $\mathcal{EL}^{++}$. Since $\mathcal{EL}^{++}$ covers many important and often used constructors (e.g., $\sqcap$, $\exists$), the saturation does already the majority of the work for many ontologies (as confirmed by our evaluation in Section 7).

In the remainder of the paper, we simply use the saturate function to denote the saturation of an initial saturation graph S, i.e., we assume that saturate creates and returns a new saturation graph S' from S, where the rules of Table 2 and 3 are exhaustively applied. We call a saturation graph S fully saturated if all the rules of Table 2 and 3 are not further applicable.

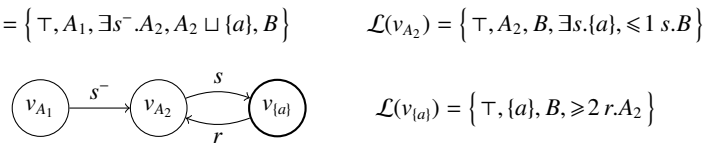

Fig. 1. Generated saturation graph for testing the satisfiability of A_1 for Example 2

3.2 Saturation Status Detection

Similarly to other saturation procedures, the presented method in Section 3.1 easily becomes incomplete for more expressive Description Logics. In order to nevertheless gain as much information as possible from the saturation, we would like to detect more precisely which nodes are critical, i.e., for which nodes the saturation was possibly incomplete. In principle, this can be easily approximated by testing for which nodes the actual tableau expansion rules are still applicable. However, since we also 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 $n \geq 1$, 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 also 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.

In the following, we use the rules of Table 4 and 5 to detect a saturation status, where incompletely handled nodes are identified and other relevant information is extracted for the assistance of the tableau algorithm. To be more precise, the rules are applied to a saturation graph and collect the nodes in the sets $\mathcal{S}_{\mathsf{o}}$, $\mathcal{S}_{!}$ and $\mathcal{S}_{?}$, where $\mathcal{S}_{\mathsf{o}}$ identifies nodes that depend on nominals, $\mathcal{S}_{!}$ identifies nodes with tight at-most restrictions, and $\mathcal{S}_{?}$ identifies critical nodes that are potentially not completely handled by the saturation. In order to facilitate the handling of at-most cardinality restrictions that are possibly not completely handled, we define the number of merging candidates as follows:

Definition 15 (Merging Candidates). *Let $S = (V, E, \mathcal{L})$ be a saturation graph and $v \in V$ a node. For a role s and a concept D, the* number of merging candidates for 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*

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

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

Table 4. Rules for detecting relevant information in the saturation graph

$\mathcal{R}_{\mathsf{o}}$-rule:	if $v \notin \mathcal{S}_{\mathsf{o}}$, and $\{a\} \in \mathcal{L}(v)$ or v has a successor node v' with $v' \in \mathcal{S}_{\mathsf{o}}$ then $\mathcal{S}_{\mathsf{o}} \longrightarrow \mathcal{S}_{\mathsf{o}} \cup \{v\}$
$\mathcal{R}_{!}$-rule:	if $v \notin \mathcal{S}_{!}$, $\leqslant m\, s.D \in \mathcal{L}(v)$, and $\sharp\mathsf{mcands}(v, s, D) = m$ then $\mathcal{S}_{!} \longrightarrow \mathcal{S}_{!} \cup \{v\}$

by the $\mathcal{R}_{!}$-rule (see Table 4) 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 $\mathcal{R}_{\mathsf{o}}$-rule adds all nodes that directly or indirectly depend on nominals to the set $\mathcal{S}_{\mathsf{o}}$, 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.

Now, the rules of Table 5 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 $\mathcal{R}_{\forall}$- and $\mathcal{R}_{\sqcup}$-rule identify nodes as critical for which the $\forall$- or the $\sqcup$-rule of the tableau algorithm is applicable. Note, for the $\mathcal{R}_{\forall}$-rule it is only necessary to check whether the concepts can be propagated to the successor nodes since the propagation to predecessors is ensured by the saturation procedure.
- The $\mathcal{R}_{\leqslant}$-rule checks for every node v whether there is a not satisfied 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, we use the $\mathcal{R}_{\mathsf{ch}}$-rule to identify nodes as incompletely handled if they have s-successor nodes, where neither D nor $\mathsf{norm}(\neg D)$ is in their label. In addition, we have to consider that the successors may have to be merged into a predecessor. Of course, this has to be checked from the perspective of the predecessors due to the reuse of nodes. Therefore, we check with the $\mathcal{R}_{\downarrow\leqslant}$ and $\mathcal{R}_{\downarrow\mathsf{ch}}$-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 as critical.
- We also need several rules for the detection of incompleteness related to nominals. First, we check with the $\mathcal{R}_{\mathsf{oo}}$-rule whether there are two nodes in the saturation graph that have the same nominal but different concepts in their label. If this is the case, then the handling of the nominals is possibly incomplete since the merging of these nodes would make all concepts on both places available. Of course, if we saturate several independent concepts in the same saturation graph, then it is not necessarily the case that all nodes with the same nominal in their label have to be merged. However, less expressive ontologies are often using nominals only in very simple ways and, therefore, a more exact analysis is usually not necessary and is often also less efficient. In addition, if a node v is nominal dependent, i.e., it has a descendant node with a nominal in its label, then other individuals can in principle propagate consequences to v. This also means that v has to be added to $\mathcal{S}_{?}$ (which is realised by the $\mathcal{R}_{\mathsf{o}?}$-rule) if there is a node for an individual, which

Table 5. Rules for detecting incompleteness in the saturation graph

$\mathcal{R}_\forall$-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	$\mathcal{S}_? \longrightarrow \mathcal{S}_? \cup \{v\}$
$\mathcal{R}_\sqcup$-rule:	if	$v \notin \mathcal{S}_?$, $C \sqcup D \in \mathcal{L}(v)$, and $\{C, D\} \cap \mathcal{L}(v) = \emptyset$
	then	$\mathcal{S}_? \longrightarrow \mathcal{S}_? \cup \{v\}$
$\mathcal{R}_\leqslant$-rule:	if	$v \notin \mathcal{S}_?$, $\leqslant m\, s.D \in \mathcal{L}(v)$, and $\sharp\mathsf{mcands}(v, s, D) > m$
	then	$\mathcal{S}_? \longrightarrow \mathcal{S}_? \cup \{v\}$
$\mathcal{R}_{\mathsf{ch}}$-rule:	if	$v \notin \mathcal{S}_?$, $\leqslant m\, s.D \in \mathcal{L}(v)$, there is an s-successor v' of v, and $\mathcal{L}(v') \cap \{D, \mathsf{norm}(\neg D)\} = \emptyset$
	then	$\mathcal{S}_? \longrightarrow \mathcal{S}_? \cup \{v\}$
$\mathcal{R}_{\downarrow\leqslant}$-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	$\mathcal{S}_? \longrightarrow \mathcal{S}_? \cup \{v\}$
$\mathcal{R}_{\downarrow\mathsf{ch}}$-rule:	if	$v \notin \mathcal{S}_?$, v' is an $\mathsf{inv}(s)$-successor of v, $\leqslant m\, s.D \in \mathcal{L}(v')$, and $\mathcal{L}(v) \cap \{D, \mathsf{norm}(\neg D)\} = \emptyset$
	then	$\mathcal{S}_? \longrightarrow \mathcal{S}_? \cup \{v\}$
$\mathcal{R}_{\mathsf{oo}}$-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	$\mathcal{S}_? \longrightarrow \mathcal{S}_? \cup \{v\}$
$\mathcal{R}_{\mathsf{o}?}$-rule:	if	$v \notin \mathcal{S}_?$, $v \in \mathcal{S}_\mathsf{o}$, and there exist some node $v_{\{a\}} \in \mathcal{S}_?$
	then	$\mathcal{S}_? \longrightarrow \mathcal{S}_? \cup \{v\}$
$\mathcal{R}_{\mathsf{o}\leqslant}$-rule:	if	$v \notin \mathcal{S}_?$, v' is an $\mathsf{inv}(s)$-successor of v, $\{a\} \in \mathcal{L}(v')$, and $\leqslant m\, s.D \in \mathcal{L}(v')$
	then	$\mathcal{S}_? \longrightarrow \mathcal{S}_? \cup \{v\}$
$\mathcal{R}_\uparrow$-rule:	if	$v \notin \mathcal{S}_?$, there is a successor v' of v, and $v' \in \mathcal{S}_?$
	then	$\mathcal{S}_? \longrightarrow \mathcal{S}_? \cup \{v\}$

could not be completely handled by the saturation. Next, we need a rule that checks interaction between nominals and at-most cardinality restrictions. The tableau algorithm applies the NN-rule for a nominal with an at-most cardinality restriction and a blockable predecessor in order to fix the number of overall neighbour nodes for this nominal. This cannot be easily handled by the saturation and, therefore, we use the $\mathcal{R}_{\mathsf{o}\leqslant}$-rule to identify such predecessors as critical by adding them to $\mathcal{S}_?$.

- Finally, the $\mathcal{R}_\uparrow$-rule marks all predecessors of critical nodes also as critical, i.e., it adds a node v to $\mathcal{S}_?$ if a successor of v is in the set $\mathcal{S}_?$.

The sets $\mathcal{S}_\mathsf{o}$, $\mathcal{S}_!$, and $\mathcal{S}_?$ are now used to define the saturation status of a saturation graph as follows:

Definition 16 (Saturation Status). *The* saturation status $\mathcal{S}$ *of a saturation graph* $S = (V, E, \mathcal{L})$ *is defined as the tuple* $(\mathcal{S}_\mathsf{o}, \mathcal{S}_!, \mathcal{S}_?)$*. We use* status *as the function that creates* $\mathcal{S}$ *from* S *by the exhaustive application of the rules of Table 4 and 5.*

Let $v \in V$ *be a node in the saturation graph* S*. With respect to* $\mathcal{S}$ *and* S*, we call* n clashed *if* $\bot \in \mathcal{L}(v)$*. Furthermore, we say* v *is* critical *if* $v \in \mathcal{S}_?$*,* nominal dependent *if* $v \in \mathcal{S}_\mathsf{o}$*, and we say* v *has* tight at-most restrictions *if* $v \in \mathcal{S}_!$*.*

A concept C is obviously unsatisfiable if its representative node v_C is clashed, whereas the satisfiability of C can only be guaranteed (for the general case) if v_C is not critical, v_C does not depend on a nominal and the knowledge base is consistent.

Consistency is explicitly required, because a concept is defined as satisfiable only if the knowledge base is consistent, which, however, cannot always be determined by the saturation procedure since it might not be able to completely handle all representative nodes for individuals. Of course, if the inconsistency of the knowledge base can be detected during the saturation, i.e., a representative node for an individual is clashed, then the $\bot$-rule propagates the unsatisfiability also to all other nodes. In contrast, if the saturation graph contains a critical node that represents an individual, then only the nominal dependent nodes are also marked as critical. Thus, for the remaining nodes, we have to require that the knowledge base is consistent in order to be able to guarantee the satisfiability of their associated concepts.

In addition, if a node is nominal dependent, then the consequences that are propagated to this node obviously depend on the concepts in the labels of the dependent representative nodes for these nominals. Therefore, we cannot generally guarantee the satisfiability of such a node without knowing the status of the representative nodes for those nominals on which v_C depends.

Please also note that a critical node for an individual makes also all nominal dependent nodes critical, which can obviously be very problematic in practice. However, we can improve the saturation graph after the initial consistency check with the tableau algorithm (see Section 5 for details) by simply replacing the node labels in the saturation graph for all individuals with the ones from the obtained completion graph. Of course, we have to distinguish deterministically and non-deterministically derived concepts in these labels, but in principle we know that they correspond to a clash-free and fully expanded completion graph and, therefore, we can consider them as not critical. Hence, the critical propagated status due to a critical node for an individual is only during the actual consistency check problematic since, afterwards, the saturation status can be updated with completely handled nodes for individuals.

Example 3. By applying the rules of Table 4 and 5 to the saturation graph that is depicted in Figure 1, we obtain $\mathcal{S}_{\mathsf{o}} = \{v_{\{a\}}, v_{A_1}, v_{A_2}\}$, $\mathcal{S}_{!} = \{v_{A_2}\}$ and $\mathcal{S}_{?} = \{v_{A_1}\}$. Note, v_{A_1} is critical in two ways. On the one hand, the concept $\exists s^{-}.B_2$ is problematic because it connects v_{A_2} with v_{A_1} for the role s. Since the at-most cardinality restriction $\leqslant 1\, s.B$ is in the label of v_{A_2} and the number of merging candidates for v_{A_2} w.r.t. s and B is 1, i.e., $\sharp\mathsf{mcands}(v_{A_2}, s, B) = 1$, every additional s-neighbour would cause merging. On the other hand, none of the disjuncts of the disjunction $A_2 \sqcup \{a\}$ is added to the node v_{A_1}. However, all nodes except v_{A_1} are completely handled by the saturation.

3.3 Correctness

It is clear that the saturation rules of Table 2 and 3 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 in an equivalently labelled node in the completion graph. Also, the termination of the saturation rules is obviously ensured since the number of nodes is limited by the concepts occurring in the knowledge base and the rules are only applied if they add new facts in the saturation graph. Analogously, the application of the rules of Table 4 and 5 for the generation of a saturation status is terminating, because

each rule application adds the node to a corresponding set (either $\mathcal{S}_?$, $\mathcal{S}_!$, 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 have to show that as long as the nodes are not identified as critical, then the presented saturation indeed infers all consequences, i.e., it is possible to extract a fully expanded and clash-free completion graph from the saturation graph. Since the tableau algorithms for the DL $\mathcal{SROIQ}$ are sound and complete, this obviously also shows the correctness of the presented saturation. Note, we also use the critical nodes for the assistance of the tableau algorithm, but we consider such nodes as possibly incomplete (e.g., by only using the inferred consequences) and, therefore, we can ignore the critical nodes for the proofs.

A direct conversion 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, we can have a node v' in the saturation graph that is an r- and an s-successor of a node v for an axiom $\mathsf{Disj}(r, s)$ in the knowledge base if s is not a super role of r. Obviously, this would be identified as a clash in the completion graph. As a consequence, it is necessary to recursively rebuild the successors with new nodes in the completion graph until we reach nominal nodes or the nodes are blocked (e.g., by ancestors with the same label). Moreover, for at-least cardinality restrictions, we have to create several successors that are stated as pairwise different with the $\dot{\neq}$ relation, whereas the same successor node is reused in the saturation graph.

To be more precise, a saturation graph $S = (V, E, \mathcal{L})$ that shows the satisfiability of a concept C can be transformed into a completion graph $G = (V', E', \mathcal{L}', \dot{\neq})$ as described below. For this, let $a_1, \ldots, a_\ell$ be all individuals in the knowledge base, Then, we initialise G with the nominal nodes $w^1, \ldots, w^\ell$ and we also create a new root node w^0 for the node v_C. Subsequently, we recursively build the successors of these nodes by using the function extr, which is defined as follows.

Definition 17 (Extraction). *Let $S = (V, E, \mathcal{L})$ be a saturation graph for the concept C w.r.t. $\mathcal{K}$ and $G = (V', E', \mathcal{L}', \dot{\neq})$ a completion graph. For $v \in V$ and $w \in V'$, the function $\mathsf{extr}^S_G(v, w)$ recursively extracts and rebuilds v and the successors of v for the node w in the completion graph with the following steps:*

1. *If there is a nominal $\{a_k\} \in \mathcal{L}(v)$ and $w \notin \{w^k, \ldots, w^\ell\}$ with $w^1, \ldots, w^\ell$ the nominal nodes for the individuals $a_1, \ldots, a_\ell$ in G, then prune w from G (i.e., remove w from V') and return.*
2. *Add every concept $C \in \mathcal{L}(v)$ to $\mathcal{L}'(w)$.*
3. *Return if w is blocked by an ancestor node of w.*
4. *For all $\exists r.\mathsf{Self} \in \mathcal{L}'(w)$, add the edge $\langle w, w\rangle$ to E' and r to $\mathcal{L}'(\langle w, w\rangle)$.*
5. *For all $\exists r.D \in \mathcal{L}'(w)$ and $\geqslant 1\, r.D \in \mathcal{L}'(w)$ for which there is a nominal $\{a_k\} \in \mathcal{L}(v_D)$ but no nominal $\{a_j\} \in \mathcal{L}(v_D)$ with $j > k$, add $\langle w, w^k\rangle$ to E' and r to $\mathcal{L}'(\langle w, w^k\rangle)$, where w^k is the nominal node for the individual a_k in the completion graph G.*
6. *For all $\exists r.D \in \mathcal{L}'(w)$, for which there is no nominal $\{a\} \in \mathcal{L}(v_D)$, create a new node w' and add the edge $\langle w, w'\rangle$ to E' and r to $\mathcal{L}'(\langle w, w'\rangle)$, then recursively call $\mathsf{extr}^S_G(v_D, w')$.*
7. *For all $\geqslant n\, r.D \in \mathcal{L}'(w)$, for which there is no nominal $\{a\} \in \mathcal{L}(v_D)$, create the new nodes $w_1, \ldots, w_n$, add the edges $\langle w, w_1\rangle, \ldots, \langle w, w_n\rangle$ to E', add r to $\mathcal{L}'(\langle w, w_1\rangle), \ldots,$*

$\mathcal{L}'(\langle w, w_n\rangle)$*, and add* $w_i \dot{\neq} w_j$ *for all* $1 \leq i < j \leq n$*, then recursively call* $\mathsf{extr}^S_G(v_D, w_k)$ *for all* $1 \leq k \leq n$.

In order to completely build the completion graph G, we call $\mathsf{extr}^S_G(v_{\{a_k\}}, w^k)$ for all individuals a_k with $1 \leq k \leq \ell$ in the knowledge base and $\mathsf{extr}^S_G(v_C, w^0)$ for the concept C with the root node w^0. Note, instead of creating new successor nodes with nominals in their label, the rebuild reuses the nominal nodes that have been added for the representation of the individual $(w^1, \ldots, w^\ell)$. This is easily possible since all nodes with the same nominal have exactly the same labels (otherwise the $\mathcal{R}_{\mathsf{oo}}$-rule would have identified the nodes as critical). In addition, if several nominals are in the label of a successor node, then the rebuild creates only an edge to those nominal node w^k with the greatest k that also has the same nominals in its label (see Step 5 of the extr function in Definition 17). Moreover, we only keep a nominal node w^k for an individual a_k if w^k does not contain a nominal $\{a_j\}$ in its label for which $j > k$ (see Step 1 of the extr function in Definition 17). Thus, each nominal occurs only in the label of one nominal node in the completion graph G.

We can now show that the expansion rules of the tableau algorithm for $\mathcal{SROIQ}$ (cf. Table 1) are not applicable to G and that G does not contain clashes for the Description Logic $\mathcal{SROIQ}$ (cf. Definition 10). Hence, we can prove the completeness of the approach.

Lemma 1 (Completeness) *Let* $S = (V, E, \mathcal{L})$ *be a fully saturated saturation graph for the concept* C *w.r.t.* $\mathcal{K}$ *and* v_C *is not critical, then a fully expanded and clash-free completion graph for* C *w.r.t.* $\mathcal{K}$ *can be extracted from* S.

Proof. As mentioned, we extract from $S = (V, E, \mathcal{L})$ for C a fully expanded and clash-free completion graph $G = (V', E', \mathcal{L}', \dot{\neq})$ that shows the satisfiability of the concept C. We initialise G, as described above, with the nominal nodes $w^1, \ldots, w^\ell$ for the individuals $a_1, \ldots, a_\ell$ occurring in $\mathcal{K}$ and with a new root node w^0 for v_C. Then, we build G by calling $\mathsf{extr}^S_G(v_C, w^0)$ and $\mathsf{extr}^S_G(v_{\{a_k\}}, w^k)$ for $1 \leq k \leq \ell$. Now, we have to show that G is clash-free and fully expanded, i.e., the tableau expansion rules are not applicable. First, let us consider the clash conditions for the completion graph (cf. Definition 10):

- Since the nodes in the saturation graph are not clashed, Clash Condition 1 is obviously not satisfied.
- Clash Condition 2 is the same as Condition 1 in the $\bot$-rule and, since we only use the labels from the saturation graph without adding additional concepts, this condition is obviously also not satisfied in the completion graph.
- Condition 2 of the $\bot$-rule is similar to the Clash Condition 3 for completion graphs and, therefore, Clash Condition 3 cannot be satisfied for blockable nodes, otherwise the nodes in the saturation graph would be clashed, which contradicts our assumption. However, since we only use one nominal node for each nominal in the completion graph, we possibly have to merge the nodes from the saturation graph that have the same nominal in their label. Thus, the extraction possibly creates new edges between nominal nodes and other nodes, but new loops can only be created by merging two nominal nodes from the saturation that are already neighbours. Hence, we have to show that we do not build a (merged) nominal node w_k which

is an r-successor of itself and $\neg\exists r.\mathsf{Self} \in \mathcal{L}'(w_k)$. However, this is excluded by Condition 3 of the $\bot$-rule.

- Analogously to Clash Condition 3, the Clash Condition 4 is not satisfied for blockable nodes (ensured by Condition 6 of the $\bot$-rule) and also not for (merged) nominal nodes (guaranteed by Condition 7 of the $\bot$-rule).
- Clash Condition 5 is obviously not satisfied in G, because we have ensured with the number of merging candidates that no at-most cardinality restriction is violated even if for every existential and at-least cardinality restriction all the successors are separately created. Note, the number of neighbour nodes can be increased for nominal nodes in the completion graph, but the $\mathcal{R}_{\mathsf{o}\leqslant}$-rule does not allow (successor) nodes with at-most cardinality restrictions as well as nominals in their label (otherwise the nodes would be identified as critical). Therefore, the nominal nodes in the completion graph for which we can add new blockable neighbour nodes does not have at-most restrictions.
- For Clash Condition 6, we have to show that there are not two nodes w and w' in the completion graph, where $\{a\} \in \mathcal{L}'(w) \cap \mathcal{L}'(w')$ and $w \dot{\neq} w'$. Since the saturation does not use the $\dot{\neq}$ relation and the extraction uses $\dot{\neq}$ only to state that the successors of at-least cardinality restrictions are pairwise different, Clash Condition 6 would only have been satisfied if the successors of al-least cardinality restrictions (with cardinalities greater than 1) had nominals in their label. However, this is already excluded by Condition 5 of the $\bot$-rule.

Next, we prove that the expansion rules of the tableau algorithm (cf. Table 1) are not applicable:

- The $\sqcap$-, $\exists$-, Self-, and $\geqslant$rules are obviously not applicable, since we have added the corresponding concepts, roles and successors already with saturation rule in the saturation graph and the subsequent extraction has added the concepts to the completion graph.
- The $\sqcup$-, $\forall$-, and ch-rules are also not applicable, because otherwise the corresponding incompleteness detection rules ($\mathcal{R}_{\sqcup}$, $\mathcal{R}_{\forall}$, $\mathcal{R}_{\mathsf{ch}}$, and $\mathcal{R}_{\downarrow\leqslant}$) would have identified the nodes as critical.
- Analogously to clashes related to at-most cardinality restrictions, the $\leqslant$rule is not applicable since the $\mathcal{R}_{\leqslant}$ and $\mathcal{R}_{\downarrow\leqslant}$-rule guarantees that every blockable node has not more merging candidates than allowed by the at-most cardinality restrictions (otherwise the node would be identified as critical). Again, since the nodes are not critical, it is also guaranteed by the $\mathcal{R}_{\mathsf{o}\leqslant}$-rule that nominal nodes (which are used as successors) do not have at-most cardinality restrictions in their labels.
- Although the saturation graph possibly has several nodes with the same nominals in their label, the o-rule is not applicable in the completion graph, because we only transferred the nodes for the individuals. Furthermore, if the nominal nodes for the individuals have also other nominals in their label, then we pruned the nominal nodes for the other individuals from the completion graph (see Step 1 of the extr function in Definition 17).
- The NN-rule is obviously also not applicable, because the $\mathcal{R}_{\mathsf{o}\leqslant}$-rule guarantees that nominal nodes, which are used as successors, do not have at-most cardinality restrictions in their labels.

Hence, the completion graph that is extracted from the saturation graph for the concept C is fully expanded and clash-free. □

4 Assisting Tableau Algorithms

In this section we present a range of optimisations to directly and indirectly assist the reasoning with tableau algorithms for the DL $\mathcal{SROIQ}$. As already mentioned, reasoning systems for more expressive Description Logics are usually very complex and they contain 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 evaluating only 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 on [3,32]. 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. More sophisticated implementations are further saving which other facts in the completion graph are the cause of the newly added ones. Hence, these dependencies between the facts in the completion graph can be used for a more exact backtracking, especially in combinations with other optimisations such as caching [28]. Thus, for new optimisation techniques that manipulate the completion graph it is important to also add these dependencies correctly, otherwise dependency directed backtracking cannot be completely supported in the presence of these optimisations.

The optimisation techniques, which 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 significantly reduced.

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 the completion graph. This improves the tableau algorithm by a faster clash detection and optimises the building of the completion graph. For example, we can directly use unsatisfiability information that is detected by the $\bot$-rule in the saturation. 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 also 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 also 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 the dependency 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 the concepts from the saturation also improves the backtracking and processing of disjunctions. Basically, the $\sqcup$-rule from the saturation extracts common (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 $\mathcal{L}(v_{A_1 \sqcup A_2}) \subseteq \{A_1 \sqcup A_2, B\}$, i.e., B is a common 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 deterministically. Hence, the backtracking does not identify the processing of alternatives of a disjunction as relevant if only such deterministic consequences are involved in the creation of a clash.

In practice, it is often wise to add the consequences from the saturation only in special situations in order to avoid that we possibly try to frequently add the same consequences. Very suitable situations are the creation of new successor nodes and the addition/processing of disjunctions. The latter one adds the common disjunct concepts, which is usually very helpful since it possibly reduces the non-determinism. Analogously, the creation of a new successor node v for an existential restriction of the form $\exists r.C$ adds C to the label of v and the addition of all other concepts from $\mathcal{L}(v_C)$ in this initialisation step enforces that the majority of the concepts are already added to its label (under the assumption that the saturation already derives the majority of all consequences for a concept). Since newly created successor nodes are initialised only once, we can add the consequences from the saturation without running into the risk that another kind of inefficiency is added to the tableau algorithm. In contrast, if several concepts, say $C_1, \ldots, C_m$, are independently added to the label of a node in separate steps (e.g., due to propagations from neighbour nodes or non-deterministic decisions) and their labels $\mathcal{L}(v_{C_1}), \ldots, \mathcal{L}(v_{C_m})$ from the saturation have a huge overlap, then we frequently try to add the same concepts to the label, which is obviously not very efficient.

The transfer of derived consequences 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. Since the transfer of derived consequences already adds the majority of all concepts, the likelihood that the rules can be applied in such an optimised way is significantly higher. Third, since the majority of all concepts are added with the transfer of derived consequences in one step, the current processing node for which the tableau algorithm applies expansion rules does not have to be changed as often as in the case where the concepts are added separately (e.g., by

separate propagations back from successor nodes). This speeds up the construction of the completion graph, because practically used tableau algorithms typically have to do some initialising tasks when switching the current processing node. Moreover, if concepts are propagated back to ancestor nodes, then it is again necessary to check whether one of these ancestor nodes is blocked before the rules in descendant nodes can be applied. Again, due to the transfer of derived consequences, the majority of all concepts that are propagated back are added in one step and, therefore, the amount of blocking tests is significantly reduced. Last but not least, the transfer of derived consequences also 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 Description Logics is pairwise blocking. However, pairwise blocking can be refined to achieve more precise blocking conditions that possibly allow for blocking earlier [11]. Since the majority of the concepts that are propagated back from successors are added by the transfer of derived consequences from the saturation, the creation of new successor nodes likely does not influence any ancestor node. As a result, the node can be blocked even before the creation and processing of successor nodes.

In principle, it would also be possible to saturate every combination of concepts that occurs in the completion graph, i.e., after adding a new concept to a label, e.g., by adding a disjunct of a disjunction, we can saturate a node that is initialised with the concepts of this label. Obviously, we would get a fast estimation whether such a combination of concepts is unsatisfiable. However, if the saturation cannot discover a clash for this combination, then the results from the saturation are not really helpful. On the one hand, we cannot use the derived consequences since we do not know the correct dependencies for them and we are not interested in saving such dependencies in the saturation procedure, because this would multiply the memory usage for the saturation graph. On the other hand, a saturated combination of concepts is possibly only required once and, therefore, it would require memory in the saturation graph although it is never used again.

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. Although the consistency of the knowledge base is necessary to guarantee the satisfiability of the saturated concepts, we can use the nodes also in the consistency check since the problematic individuals are also tested in this step. 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 the processing of them, 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 $\mathsf{norm}(\neg C)$ 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 at least one additional neighbour before some nodes have to be merged. In principle, it would be also possible to detect more precisely whether there is a thigh at-most restriction for a role that is in the label of the edge between v and its ancestor. However, this would make the generation of the successors dependent on the roles in this edge label, which is much more problematic for implementations since roles can be added to such an edge label without influencing the node labels and, therefore, possibly many and more complicated checks are necessary to decide whether the creation of successor can be blocked or has to be reactivated after the roles have changed. Thus, only checking whether v' has tight at-most restrictions is a simplification, which is, for many cases, sufficient and cannot negatively influence the performance due to costly checks.

Since nodes can easily have a large number of successor node (e.g., due to at-least cardinality restrictions with big cardinalities), the blocked creation of 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

In 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 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 [4] or known/possible set classification [6]. Therefore, these optimised classification algorithms use an intelligent testing order 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 be syntactically found in the ontology axioms, and, typically, the more told subsumers can be extracted, the bigger is the benefit for the classification algorithms. However, a more detailed extraction of told subsumers from the 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 classification algorithm since also (some) semantic consequences are considered by the saturation. As new and more accurate told subsumers, we can simply use all the classes in $\mathcal{L}(v_A)$ for each class A that has to be classified. 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. It is not even relevant whether v_A depends on nominals or not, because subsumers and possible

subsumers can be extracted from a satisfiability test, where nominals are not further influenced by additional assertions (like it is the case for instance tests, where we have to check whether an individual is an instance of a specific concept). If all nodes of an ontology are not critical, we already get all subsumers from the saturation and, therefore, only a transitive reduction is necessary to build the class hierarchy. Thus, with a preceded saturation we automatically get a one-pass classification for simple ontologies.

4.3 Model Merging

Many ontologies contain axioms of the form $C \equiv D$, which can be seen as an abbreviation for $C \sqsubseteq D$ and $D \sqsubseteq C$. Treating axioms of the form $A \equiv D$ with A an atomic concept as $A \sqsubseteq D$ and $D \sqsubseteq A$ can, however, downgrade the performance of tableau algorithms since absorption might not apply to $D \sqsubseteq A$, i.e., the axiom is internalised into $\top \sqsubseteq \mathsf{norm}(\neg D \sqcup A)$. 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 $\mathsf{norm}(\neg D)$ (exploiting that $D \sqsubseteq A$ is equivalent to $\neg A \sqsubseteq \mathsf{norm}(\neg D)$) [13].[3] Unfortunately, using such an unfolding rule also comes at a price since the tableau algorithm is no longer forced to add either A or $\mathsf{norm}(\neg D)$ to each node in the completion graph, i.e., we might not know for some nodes whether they represent instances of A or $\neg A$. This means that we cannot exclude A as possible subsumer for other (atomic) concepts if the nodes in the completion graph do not contain A, which is an important optimisation for classification procedures.

To compensate this, we can create a "candidate concept" A^+ for A, for example by partially absorbing D, which is then automatically added to a node in the completion graph if the node is possibly an instance of A. Hence, if A^+ is not added to a node label, then we know that A is not a possible subsumer of the concepts in the label of this node.

Although the candidate concepts already allow a significant pruning of subsumption tests, there are still ontologies where we have to add these candidate concepts to many node labels, especially if only a limited absorption of D for an $A \equiv D$ axiom is possible. Hence, A can still be a possible subsumer for many concepts. The saturation graph can, however, again be used to further improve the identification of (more or less obvious) non-subsumptions. Basically, if a candidate concept A^+ for $A \equiv D$ is in the label of a node v in the completion graph, then we test whether the merging with the saturated node $v_{\mathsf{norm}(\neg D)}$ is possible. Since D is often a conjunction, we can also try to merge v with the representative node for a disjunct of $\mathsf{norm}(\neg D)$. If the "models" can be "merged" as defined below, then v is obviously not an instance of A.

Definition 18 (Model Merging). *Let $S = (V, E, \mathcal{L})$ be a fully saturated saturation graph and $G = (V', E', \mathcal{L}', \dot{\neq})$ be a fully expanded and clash-free completion graph for a knowledge base $\mathcal{K}$. A node $v \in V$ is* mergeable *with a node $w \in V'$ if*

- *v is not critical, not nominal dependent, and not clashed,*
- *$\mathcal{L}(v) \cup \mathcal{L}'(w)$ does not contain $\{C, \mathsf{norm}(\neg C)\}$,*

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

- $\mathcal{L}(v) \cup \mathcal{L}'(w)$ *does not contain the concepts* A_1 *and* A_2 *such that* $(A_1 \sqcap A_2) \sqsubseteq C \in \mathcal{K}$ *and* $C \notin (\mathcal{L}(v) \cup \mathcal{L}'(w))$*,*
- *w is not an r-neighbour of w for a concept* $\neg\exists r.\mathsf{Self} \in \mathcal{L}(v)$*,*
- *v is not an r-successor of v for a concept* $\neg\exists r.\mathsf{Self} \in \mathcal{L}'(w)$*,*
- *if* $\forall r.C \in \mathcal{L}(v)$ $(\leqslant m\, r.C \in \mathcal{L}(v))$*, then* $C \in \mathcal{L}'(w')$ $(\mathsf{norm}(\neg C) \in \mathcal{L}'(w'))$ *for every r-neighbour* w' *of w;*
- *if* $\forall r.C \in \mathcal{L}'(w)$ $(\leqslant m\, r.C \in \mathcal{L}'(w))$*, then* $C \in \mathcal{L}(v')$ $(\mathsf{norm}(\neg C) \in \mathcal{L}'(v'))$ *for every r-successor* v' *of v.*

Obviously, the conditions that guarantee that the models are mergeable can be checked very efficiently. Of course, it is also possible to relax some of the conditions. 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 completion graph does not use nominals. Moreover, if there are concepts in the completion graph that possibly have an interaction with the merged node in saturation graph, then we can extend the saturation with a new node, where also these concepts are considered, and test the model merging again. For instance, if a node w in the completion graph is not mergeable with a node v in the saturation graph due to an r-successor v' of v and a concept $\forall r.C$ in the label of w for which $C \notin \mathcal{L}(v')$, then we can saturate a new node v'' with $\mathcal{L}(v'') \supseteq \mathcal{L}(v) \cup \{\forall r.C\}$ and check whether v'' is mergeable. Especially with this extension, we can identify many non-instances for nodes in the completion graph also for more expressive ontologies.

In contrast, if there are interactions in the completion graph with concepts from the saturation, then it is often not easily possible to extend the model merging approach such that the non-subsumption can be guaranteed. Basically, we do not want to modify 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, is non-trivial since we have to exclude interactions with already tested nodes. For example, if a node w in the completion graph is not mergeable with a node v in the saturation graph due to an r-neighbour w' of w 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 exclude that the merging of w' with v_C causes new consequences that are propagated back to w, which is especially non-trivial if there are several universal restrictions in the label of v that would affect w'.

Note, although other proposed (pseudo) model merging techniques [8] work in principle very similar, there are also some significant differences. For example, the presented merging test is only applied if corresponding candidates concepts are in the label of nodes, which already reduces the number of tests. In addition, we test the merging against the saturation graph and, therefore, we do not have any significant overhead. In contrast, for other approaches it is often necessary to build separate completion graphs for those concepts to which the model merging must be tested. Moreover, the presented approach is easily applicable to very expressive Description Logics such as $\mathcal{SROIQ}$. Admittedly, very expressive Description Logics may produce more critical nodes, nevertheless it is not necessary to deactivate the model merging if certain language features

are used and, thus, this model merging approach has a very good pay-as-you-go behaviour.

5 Saturation Improvements

Obviously, the assistance of the tableau algorithm with the saturation works better when as few nodes as possible are marked as 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 as critical. In the following, we present different approaches how the saturation can be improved such that the number of critical nodes can be reduced. As a result, a better assistance of the tableau algorithm is possible.

5.1 Extending Saturation to more Language Features

Obviously, the presented saturation procedure can be extended to more language features. For example, to completely cover the DL $\mathcal{EL}^{++}$, it would be necessary to integrate a more sophisticated handling of nominals into the presented saturation procedure. In principle, this is possible by saving, 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 save for v that the nominal $\{a\}$ is used by this descendant. If we find a node v, where we have saved that a nominal $\{a\}$ is used by several descendant nodes, say $v^1, \ldots, v^n$, then we create a new node u where the labels of $v^1, \ldots, v^n$ are merged, and we "reproduce" the paths of predecessors from the merged nodes up to v such that the new consequences can also be propagated to v. To be more precise, if $w_1, \ldots, w_m$ represent the path of predecessors for one merged node $w \in \{v^1, \ldots, v^n\}$ up to the node v, then we copy these nodes to new nodes $w'_1, \ldots, w'_m$ such that they represent a new path of predecessors from the new node u up to v. Obviously, we only have to reproduce predecessors if they are influenced by new consequences from the new node with the merged label. Furthermore, if a nominal is used by a node v and by a descendant v' of v, then we add all concepts from the label of v' to v and we again reproduce the path of all predecessors from the descendant node v' to v for the node v. Thus, this approach has still a very good pay-as-you-go behaviour if nominals are used as allowed by the OWL 2 EL profile. However, as of now, EL ontologies often only use nominals in much simpler ways for which the presented saturation procedure is often still sufficient.

As known from literature, the saturation can also be extended to more expressive Horn Description Logics, e.g., Horn-$\mathcal{SHIQ}$ [16] or even Horn-$\mathcal{SROIQ}$ [21]. Although such extended saturation procedures are obviously very efficient for ontologies in these fragments, it is not clear how they perform for other ontologies, for example, in combination with 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. However, some of the features of these Horn-languages can easily be supported and, in practical implementations, we can simply limit the number of the

nodes that are processed by the saturation and by marking the remaining nodes as critical. For example, it is very interesting to consider the propagation of new concepts to successors for universal restrictions of the form $\forall r.C$. Of course, we are not allowed to directly modify existing r-successors, but we can easily create a new r-successor that we extend by the propagated concept C. Furthermore, the previous r-successor has to be removed from a node v such that the incompleteness detection rule $\mathcal{R}_\forall$ for the concept $\forall r.C$ does not mark the node v as critical, which is obviously not the case if now all newly connected r-successors also include the concept C and are completely handled. Note, the extension of nodes can be implemented 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 node that has to be extended, a mapping for the concepts, for which the node has to be extended, to other nodes, whereby we can save and reuse the extended nodes. Thus, if several predecessors propagate the same concepts to the same successors, then we create a node with the corresponding extension only once. Obviously, this approach can further be improved by using suitable orders for the nodes and the concepts that have to be used for the extensions.

In principle, the support for at-most restrictions of the form $\leqslant 1r.\top$ is analogously possible. At least, the labels of the 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'', we reproduce v as a successor of v'' and we connect v'' as an r-successor of the node that we have reproduced for v. Again, the reproducing of v is necessary, because we are not allowed to directly modify the successors, which would, however, be the case if we connect v'' as an r-successor of v. Thus, Horn-$\mathcal{SHIF}$ can be almost completely supported with rather small extensions of the presented saturation procedure.

Saturation procedures can further be extended beyond Horn Description Logics, for instance, for the DL $\mathcal{ALCH}$ [22]. Therefore, they also have to handle non-determinism which is, for example, caused by disjunctions. Since the dependencies are usually not tracked by the saturation algorithm, such extensions can easily become inefficient when all alternatives of concepts have to be saturated. Although this approach still works well for a range of ontologies, it is also possible to construct examples, where this approach is no longer efficient. Hence, it is possibly better to keep the basic saturation algorithm deterministic and to process the remaining parts with the tableau algorithm, which integrates several optimisations (e.g., semantic branching, boolean constant propagation, dependency directed backtracking, caching) to efficiently handle non-determinism.

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 are intensively using nominals. Analogously, if there are few nodes with incompletely handled concepts (e.g., disjunctions) and these nodes are referenced by many other nodes,

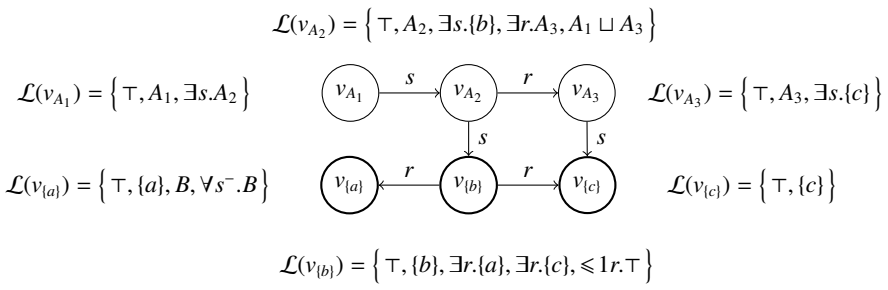

Fig. 2. Saturation graph for testing the satisfiability of A_1

then also all these other nodes are 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 4. Let us assume that the TBox $\mathcal{T}_4$ 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 $\mathcal{T}_4$, 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 $\mathcal{S} = (\mathcal{S}_\mathsf{o}, \mathcal{S}_!, \mathcal{S}_?)$, where $v_{\{b\}}$ has a tight at-most restrictions, i.e., $\mathcal{S}_! = \{v_{\{b\}}\}$, 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 for a better support of the features of more expressive Description Logics (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 Description Logics 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 update the saturation graph with results from fully expanded and clash-free completion graphs that are generated for consistency or satisfiability checks. Roughly speaking, we replace the labels of critical nodes in the saturation graph with corresponding labels from the completion graphs, for which we know that they are completely handled by the tableau algorithm. Then, we apply the saturation rules again and we update the saturation status, which hopefully results in an improved saturation graph with less critical nodes. However, since the completion graph contains deterministically and non-deterministically derived concepts, we also have to distinguish them for the saturation. An interesting way to achieve this is to simultaneously manage two saturation graphs: one where only the deterministically derived concepts are added and a second one where also the non-deterministically derived concepts and consequences are considered. If the non-deterministic consequences have only a locally limited influence, i.e., the non-deterministically added concepts propagate new consequences only to a limited number of ancestor nodes, then, by comparing both saturation graphs, we can possibly identify not further influenced ancestor nodes, which can then be considered as not critical. Thus, this approach allows to further improve the construction of new completion graphs by transferring new and more results from the updated saturation. In the following, we present this approach in a way, where the saturation graphs for the deterministic and the non-deterministic consequences are implicitly managed as the extensions of the original saturation graph, which simplifies the update process.

First, we define a saturation patch, which is the data structure for managing the information that is necessary for the extension of a saturation graph.

Definition 19 (Saturation Patch). *Let* $\mathsf{fclos}(\mathcal{K})$ *(*$\mathsf{Rols}(\mathcal{K})$*) denote the concepts (roles) that possibly occur in the completion graph for the knowledge base* $\mathcal{K}$ *as defined in Definition 8. A* saturation patch *P for a saturation graph* $S = (V, E, \mathcal{L})$ *w.r.t.* $\mathcal{K}$ *is a tuple* $P = (P_V, P_{\mathcal{L}d}, P_{\mathcal{L}n}, P_{mc}, P_{nd})$*, where*

- $P_V \subseteq V$ *denotes the set of patched nodes in the saturation graph,*
- $P_{\mathcal{L}d} \colon P_V \to 2^{\mathsf{fclos}(\mathcal{K})}$ *is the mapping of patched nodes to a set of deterministically derived concepts,*
- $P_{\mathcal{L}n} \colon P_V \to 2^{\mathsf{fclos}(\mathcal{K})}$ *is analogously the mapping of patched nodes to a set of non-deterministically derived concepts,*
- $P_{mc} \colon P_V \times \mathsf{Rols}(\mathcal{K}) \times \mathsf{fclos}(\mathcal{K}) \to \mathbf{N}_0$ *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*
- $P_{nd} \subseteq P_V$ *denotes the patched nodes that are nominal dependent.*

Given two saturation patches $P = (P_V, P_{\mathcal{L}d}, P_{\mathcal{L}n}, P_{mc}, P_{nd})$ *and* $P' = (P_V', P_{\mathcal{L}d}', P_{\mathcal{L}n}', P_{mc}', P_{nd}')$*, the saturation patch* $P \circ P'$ *is defined as the tuple consisting of*

- $P_V \cup P_V'$,
- $P_{\mathcal{L}d} \cup \{v \mapsto C \mid v \mapsto C \in P_{\mathcal{L}d}' \text{ and } v \notin P_V\}$,
- $P_{\mathcal{L}n} \cup \{v \mapsto C \mid v \mapsto C \in P_{\mathcal{L}n}' \text{ and } v \notin P_V\}$,
- $P_{mc} \cup \{\langle v, s, C\rangle \mapsto n \mid \langle v, s, C\rangle \mapsto n \in P_{mc}' \text{ and } v \notin P_V\}$, *and*
- $P_{nd} \cup (P_{nd}' \setminus P_V)$.

Obviously, a saturation patch $P = (P_V, P_{\mathcal{L}d}, P_{\mathcal{L}n}, P_{mc}, P_{nd})$ has to identify the nodes that have to be patched, which is realised with the set P_V. Moreover, we have mappings for deterministic ($P_{\mathcal{L}d}$) and non-deterministic ($P_{\mathcal{L}n}$) sets of concepts that describe how these nodes can be extended such that they do not have to be considered as critical. We also have to store the number of merging candidates (P_{mc}) and the nominal dependency (P_{nd}) for the patched nodes, because this information is required for the generation of an updated saturation status. Note, although we distinguish deterministically and non-deterministically derived concepts, we consider the mapping for the number of merging candidates and the mapping for nominal dependency 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 [11]. 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. Basically, we have to consider the nominal dependency as non-deterministic information since the labels do not have to be identical and 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 constructed completion graphs for satisfiability and consistency tests are very suitable for the extraction of patches. For instance, if the tableau algorithm creates a fully expanded and clash-free completion graph for testing the satisfiability of a concept C, then a patch for the node v_C can be extracted from the root node for which the concept C is asserted. In contrast, the completion graph of the 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.

In principle, it would be possible to extract patches also from other nodes, but this would require a more detailed analysis of the completion graph. 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 ancestor 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. However, if it is difficult to extract a certain kind of information, then we simply restrict the patch creation appropriately.

Note that we can combine several patches into one with the $\circ$-operator. However, if both patches contain information about the same node, then we keep the information for

this node only from one patch instead of mixing the information. Thus, the information from the other patches gets lost for all common nodes.

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

Definition 20 (Patch Application). *Let $S = (V, E, \mathcal{L})$ be a saturation graph and $P = (P_V, P_{\mathcal{L}d}, P_{\mathcal{L}n}, P_{mc}, P_{nd})$ a saturation patch for S. We call $S[P]$ the* application *of the patch P to the saturation graph S.*

The deterministic saturation graph *that is associated with a patch application $S[P]$ is the fully saturated saturation graph $S' = \mathsf{saturate}((V, E, \mathcal{L}'))$, where $\mathcal{L}'$ is the extension of $\mathcal{L}$ by the concepts from $P_{\mathcal{L}d}$, i.e., $\mathcal{L}'(v) = \mathcal{L}(v) \cup P_{\mathcal{L}d}(v)$ for each $v \in E$.*

Let $S' = (V', E', \mathcal{L}')$ be the deterministic saturation graph that is associated with $S[P]$, where $P = (P_V, P_{\mathcal{L}d}, P_{\mathcal{L}n}, P_{mc}, P_{nd})$, then the non-deterministic saturation graph *that is associated with $S[P]$ is the saturation graph $S'' = \mathsf{saturate}^{\setminus P_V}((V', E', \mathcal{L}''))$, where $\mathcal{L}''$ is the extension of $\mathcal{L}'$ by the concepts from $P_{\mathcal{L}n}$, i.e., $\mathcal{L}''(v) = \mathcal{L}'(v) \cup P_{\mathcal{L}n}(v)$ for each $v \in V'$, and $\mathsf{saturate}^{\setminus V}$, with the set of nodes V, is a modified version of the* saturate *function such that only the $\forall$-rule is applied to the nodes in V and the $\forall$-rule does not propagate new concepts to nodes in V.*

In order to extend a patch application $S[P]$ with a new patch P', we combine the patches such that we keep all information from the new patch, i.e., $S[P' \circ P]$. This allows for updating already patched nodes, whereby the associated saturation graphs can be further improved. Each patch application $S[P]$ implicitly represents a deterministic saturation graph, where only the deterministic consequences from the applied patch P are considered in an extension of S, and a non-deterministic saturation graph, where also the non-deterministic consequences are considered.

Note, in order to saturate the non-deterministic saturation graph, we use the slightly modified $\mathsf{saturate}^V$ function, where the modification of the patched nodes by the saturation rules is avoided. The standard saturation rules possibly propagate new consequences also to patched nodes, which is unfavourable if these consequences are derived from the processing of different non-deterministic alternatives. Basically, if a node v and an ancestor of v are patched, then the standard saturation rules might propagate new consequences that are obtained from the patching of v up to the ancestors. If the 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 also considered to be non-deterministic, this does not produce incorrect results. In order to, nevertheless, avoid such unfavourable interactions, we use the modified saturation $\mathsf{saturate}^{\setminus V}(S')$ 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 $v \in V$. Although the saturation rules can possibly also influence patched nodes in the

deterministic saturation graph, this is not problematic since all consequences are indeed deterministic consequences.

Note, for practical implementations, it is not necessary to rebuild the deterministic and non-deterministic saturation graphs from scratch after the application of new patches. In principle, we can incrementally update the last created versions of the deterministic and non-deterministic saturation graphs, which is especially trivial for the deterministic one since all information from new patches can be used to extend the previous version of the graph. Furthermore, we can obviously reuse the data of the previous saturation graph for all nodes that are not influenced by the patches. Also, since the non-deterministic consequences only influence ancestor nodes, we can use the last created non-deterministic saturation graph, apply the new patches, and then we successively update the ancestors of the newly patched nodes until we reach ancestors that are not further influenced by consequences of the new patches, i.e., their labels are identical to the labels of the corresponding nodes in the last non-deterministic saturation graph. Hence, also the construction of the non-deterministic saturation graph is usually not a significant overhead, especially if the non-deterministically added consequences only have a locally limited influence. In practice, it can be useful to further limit the number of ancestors that are updated for the new non-deterministic consequences. If the limit is reached, then the remaining ancestors are simply marked as critical. This allows for guaranteeing a limited overhead for the management of the non-deterministic saturation graph.

In order to further enable the usage of patched saturation graphs for the assistance of the tableau algorithm, e.g., for the transfer of result into the 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 use a modified $\mathsf{status}^{\setminus V}$ function instead of status, which applies the rules of Table 4 and 5 only to nodes that are not in V. Obviously, we also have to use a modified $\sharp\mathsf{mcands}'$ function in $\mathsf{status}^{\setminus V}$ since we have to use the information from the patch for the patched nodes. To be more precise, when P_{mc} denotes the mapping to the number of merging candidates in the considered patch, then $\sharp\mathsf{mcands}'(v, s, D)$ has to return $P_{mc}(\langle v, s, D\rangle)$ if v is a patched node, and $\sharp\mathsf{mcands}(v, s, D)$ otherwise.

In addition, we have to initialise the sets $\mathcal{S}_!$, $\mathcal{S}_\mathsf{o}$, and $\mathcal{S}_?$ for the patched nodes with the information from the applied patch. For this, we use the $\mathsf{initStatus}$ function (cf. Algorithm 1), which is parametrised by the saturation patch, for which we create the initial status, and the *deterministic* flag that determines whether the status is build for the deterministic or non-deterministic saturation graph. Basically, if for a patched node is known that it has a tight at-most restriction, then $\mathsf{initStatus}$ adds this node to $\mathcal{S}_!$. Analogously, the $\mathsf{initStatus}$ function adds a patched node v to $\mathcal{S}_\mathsf{o}$ if the patch contains the information that v is nominal dependent, i.e., $v \in P_{nd}$. The patched nodes in the non-deterministic saturation graph are obviously not critical since their labels are directly extracted from fully expanded and clash-free completion graphs, i.e., we have already shown that these nodes are satisfiable. However, in the deterministic saturation graph we consider only deterministic consequences and, therefore, we have to mark all patched nodes, which are also associated with non-deterministically consequences, as

Algorithm 1 initStatus(P, *deterministic*)

Output: Creates and returns an initial status for the patch P

1: $(P_V, P_{\mathcal{L}d}, P_{\mathcal{L}n}, P_{mc}, P_{nd}) \leftarrow P$
2: $(\mathcal{S}_!, \mathcal{S}_o, \mathcal{S}_?) \leftarrow (\emptyset, \emptyset, \emptyset)$
3: **for** $v \in P_V$ **do**
4: **if** there exists $\leqslant m\, r.C \in (P_{\mathcal{L}d} \cup P_{\mathcal{L}n})$ such that $P_{mc}(\langle v, r, C\rangle) = m$ **then**
5: $\mathcal{S}_! \leftarrow \mathcal{S}_! \cup \{v\}$
6: **end if**
7: **if** $v \in P_{nd}$ **then**
8: $\mathcal{S}_o \leftarrow \mathcal{S}_o \cup \{v\}$
9: **end if**
10: **if** *deterministic* = *true* **and** $P_{\mathcal{L}n}(v) \neq \emptyset$ **then**
11: $\mathcal{S}_? \leftarrow \mathcal{S}_? \cup \{v\}$
12: **end if**
13: **end for**
14: **return** $(\mathcal{S}_!, \mathcal{S}_o, \mathcal{S}_?)$

critical, i.e., these nodes are added to the set $\mathcal{S}_?$ if *deterministic* is set to *true*. In order to generate the actual saturation status for the deterministic and non-deterministic saturation graphs, we initialise the status sets by calling initStatus for the applied patch and the corresponding value for the parameter *deterministic*. Then, the function status$^{\setminus P_V}$ is called with the initial saturation status and the corresponding saturation graph, where P_V denotes the set of patched nodes in the applied patch.

Analogously to the deterministic and non-deterministic saturation graphs, every new saturation status can be incrementally 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 assisting of tableau algorithms. In order to describe the changes and enhancements for the transfer of saturation results into completion graphs, let $(\mathcal{S}^d_!, \mathcal{S}^d_o, \mathcal{S}^d_?)$ $((\mathcal{S}^n_!, \mathcal{S}^n_o, \mathcal{S}^n_?))$ denote the saturation status of the deterministic (non-deterministic) saturation graph S^d (S^n) of a patch application $S[P]$. Obviously, we can still directly add inferred consequences from a node v_C of the deterministic saturation graph S^d to a node v in the completion graph if C is in the label of v. Also, the processing of successors can still be blocked if the node v in the completion graph has a label that is identical to the label of a node v' in the deterministic saturation graph S^d and v' is neither critical nor nominal dependent, i.e., $v' \notin (\mathcal{S}^d_? \cup \mathcal{S}^d_o)$. In addition, we can now also search v' in the non-deterministic saturation graph S^n. Thus, the processing can also be blocked if there are descendant nodes that cannot be completely handled by the saturation, e.g., descendants for which non-deterministic rule applications are necessary, but the (ancestor) nodes that are used for blocking are not further critical in the non-deterministic saturation graph S^n due to the applied patches. Obviously, it is very beneficial when patches add non-deterministic consequences that have only a local impact, i.e., the patches propagate only concepts to a certain number of ancestors, and, then, the labels in the deterministic and non-deterministic saturation

graphs are identical. We know for these ancestors that they are not further influenced by non-deterministic consequences and, therefore, they can also be directly used for blocking. In particular, since the labels of these ancestor nodes are identical to corresponding labels in the deterministic saturation graph, we can establish the blocking analogously to the blocking with nodes from the deterministic saturation graph S^d, i.e., already with the creation and initialisation of new nodes in the completion graph, which makes an explicit search for a blocker node unnecessary.

We still have the unfavourable condition that the nodes in the saturation graph are not allowed to be nominal dependent for the blocking. However, also this restriction can be relaxed such that it works 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 fully expanded and clash-free completion graph. Furthermore, we ensure, on the one hand, that each subsequent saturation patch is also 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 to 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 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 on, 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 on, then we only have to reactivate the processing of this node if the nodes for the corresponding individuals are expanded differently. Note, this approach also keeps the patching of the saturation graphs consistent with respect to the individuals that are used as nominals.

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 with the non-deterministic extension $\{\forall s.D, \forall t.D\}$.

At least, this strategy ensures 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 "solved" 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 thigh at-most restrictions by trying to reducing the number of such nodes.

Example 5. 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. the TBox $\mathcal{T}_4$, 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 together. From this completion graph, we extract an initial saturation patch P^1 for the individuals, i.e., $P^1 = ({P_V}^1, {P_{\mathcal{L}d}}^1, {P_{\mathcal{L}n}}^1, {P_{mc}}^1, {P_{nd}}^1)$ with ${P_V}^1 = \{v_{\{a\}}, v_{\{b\}}, v_{\{c\}}\}$, ${P_{\mathcal{L}d}}^1 = \{v_{\{a\}} \mapsto \{\{c\}\}, v_{\{c\}} \mapsto \{\{a\}, B, \forall s^-.B\}\}$, ${P_{\mathcal{L}n}}^1 = \emptyset$, ${P_{mc}}^1 = \{\langle v_{\{b\}}, r, \top\rangle \mapsto 1\}$, and ${P_{nd}}^1 = \{v_{\{a\}}, v_{\{b\}}, v_{\{c\}}\}$. Note, although the nodes for the individuals a and c are merged in the completion graph, we have to patch $v_{\{a\}}$ and $v_{\{b\}}$ 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 ${P_{\mathcal{L}n}}^1$. However, for ease of presentation, we omitted uninteresting patch data and we simply used $\emptyset$ for ${P_{\mathcal{L}n}}^1$.

The deterministic saturation graph that is obtained from the application of P^1 to our initial saturation graph is 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 $\{\top, A_3, \exists s.\{c\}, B, \forall s^-.B\}$. Since the patch does not contain non-deterministic information, the non-deterministic saturation graph that is associated with the application of P^1 is identical to the deterministic one.

Now, the saturation status for the these saturation graphs 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}. Hence, we initialise a new completion

graph with a node v, for which the concept A_1 is asserted, in order to test the satisfiability of A_1 with the tableau algorithm. Since the disjunction will also be added to this completion graph for an s-successor v' of v, the tableau algorithm has to choose between the disjuncts A_1 and A_3. Obviously, A_1 is satisfiable, 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 propagate B to the label of v and, thus, we could create a patch $P^2 = (\{v_{A_1}\}, \emptyset, \{v_{A_1} \mapsto \{B, \forall s^-.B\}\}, \emptyset, \{v_{A_1}\})$. Obviously, the node v_{A_1} is also with the application of P^2 in the associated deterministic saturation graph critical and, therefore, its usage for assistance (e.g., blocking in new completion graphs) is limited. In contrast, if A_1 is non-deterministically added to v', then we can extract a saturation patch $P^3 = (\{v_{A_1}\}, \emptyset, \emptyset, \emptyset, \{v_{A_1}\})$ and by applying P^3, we can remove the critical mark for the node v_{A_1} also in the associated 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 have only considered patching from fully expanded and clash-free completion graphs. Of course, it is also possible to integrate unsatisfiability information form completion graph into the saturation graphs. 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 $\bot$. The management of unsatisfiable concepts in the saturation graph has the significant benefit that $\bot$ is also propagated to other nodes and we can immediately identify many other unsatisfiable concepts.

Note, especially with the extraction and application of patches, the assistance of the tableau algorithm with the saturation graphs also integrates a very intelligent caching technique into the tableau-based reasoning system. Of course, the realised caching is limited to certain nodes that are not further influenced by predecessors, but it also works with nominals and inverse roles. Moreover, it is very fast and can automatically propagate unsatisfiability and satisfiability statuses to other concepts and labels.

The non-deterministic saturation graph can also be used to further improve the classification algorithm. As hitherto, all classes in the label of v_A in the deterministic saturation graph are indeed subsumers of the class A and if v_A is not critical, then $\mathcal{L}(v_A)$ contains all subsumers. Now, if the 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 and, in particular, if the label is also the same as in the deterministic saturation graph, then we again know that $\mathcal{L}(v_A)$ contains all subsumers.

6 Related Work

There are already some approaches that combine the reasoning techniques of fully-fledged Description Logic reasoners with specialised procedures for specific fragments. For instance, the reasoning system MORe [1] uses module extraction to extract a part of an ontology that can be completely handled with a more efficient reasoning system and

the fully-fledged reasoner is then used for the remaining parts of the ontology. Note, our approach works more from the opposite direction: we apply the saturation and simply ignore not supported features and then we detect which parts cannot be completely handled. Since MORe uses other reasoners as black-boxes, it is, in principle, also 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 combination.

In comparison with our approach, the technique in MORe is much more general and flexible. For example, it is easily possible to exchange the fully-fledged reasoner in MORe with a reasoning system for which it is know that it works better for certain kinds of ontologies. In addition, it is not necessary to modify existing reasoners, whereas our approach has to be implemented into a reasoning system that satisfies certain requirements (e.g., binary absorption). Moreover, our approach has a significant higher implementation effort since the saturation and the tableau algorithm has to operate on compatible data structures, which usually means that an appropriate saturation algorithm has to be implemented into a tableau-based reasoning system.

However, our approach 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 to fix critical parts in the saturation, whereby not supported 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 not supported features. Thus, our approach result in a better pay-as-you-go behaviour. Moreover, we can use intermediate results from the saturation, whereas the technique in MORe uses the externally provided interface 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. For instance, 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 the 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 ad-

vantage for ontologies that are almost completely in the efficiently supported fragment since our approach does not have a similar overhead for such ontologies.

Another reasoning system that combines different reasoning techniques is WSClassifier [25], 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 WSClassifier, 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.

The advantages and disadvantages of our approach are in principle similar as in the comparison with the technique realised in MORe. However, the approach of WSClassifier 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 can possibly not be used for all reasoning tasks. If such a simplification is, however, applicable, then it often improves the reasoning performance for corresponding ontologies.

7 Implementation and Evaluation

We extended Konclude[4] [29] 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 $\mathcal{SROIQ}$ [9] with extensions for the handling of nominal schemas [26]. Konclude integrates many state-of-the-art optimisations such as lazy unfolding, dependency directed backtracking, caching, etc. Moreover, Konclude uses partial absorption [27] in order to significantly reduce the non-determinism in ontologies, which makes Konclude very suitable for the integration of saturation procedures.

Our integration of the saturation algorithm in Konclude almost covers the Description Logic Horn-$\mathcal{SRIF}$ by using the saturation extensions described in Section 5.1 such that universal restrictions that propagate concepts to successors and the merging of successors due to functional at-most restrictions can be handled (basically only the merging with predecessors is not integrated). The number of nodes that are additionally processed for the handling of these saturation extensions is mainly limited by the number of concepts occurring in the knowledge base. However, the saturation in Konclude only supports a very limited handling of ABox individuals since the ABox individuals also have to be handled by the tableau algorithm (at least in the worst-case) and several representations of the ABox individuals easily multiply the memory consumption.

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

Table 6. Statistics of ontology metrics for the evaluated ontology repositories (Ø stands for average and M for median)

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

Table 7. Ontology metrics for selected benchmark ontologies

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
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
OBI	$\mathcal{SHOIN}$	32,157	3,533	84	160

To counteract this, 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 coping 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 effect of Konclude's integrated saturation procedure extended by the presented optimisation for the assistance of the fully-fledged tableau algorithm. In addition, we compare the results of

Konclude to the other state-of-the-art reasoners FaCT++ 1.6.2 [31], HermiT 1.3.8 [7], MORe 0.1.6 [1], and Pellet 2.3.1 [23]. The evaluation uses a large test corpus of ontologies which have been obtained by collecting all downloadable and parseable ontologies from the Gardiner ontology suite [5], the NCBO BioPortal,[5] the National Cancer Institute thesaurus (NCIt) archive,[6] the Open Biological Ontologies (OBO) Foundry [24], the Oxford ontology library,[7] the TONES repository,[8] and those subsets of the OWLCorpus [19] that were gathered by the crawlers Google, OntoCrawler, OntoJCrawl, and Swoogle.[9] 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). Since Konclude does not yet support datatypes, we removed all data properties and we replaced all data property restrictions with *owl*:*Thing* in all ontologies. Table 6 shows an overview of our obtained test corpus with overall 6,141 ontologies including statistics of ontology metrics for the source repositories. Please note that 34.9 % of all ontologies are not even in the OWL 2 DL profile, which is, however, mainly due to undeclared entities.

In addition to our test corpus, we present results for explicitly selected ontologies (shown in Table 7) which are frequently used in other evaluations. This enables a concreter comparison and to directly show the effect of our approach for well-known benchmark ontologies. In particular, the upper part of Table 7 consists of the well-known OWL 2 EL ontologies Gazetteer and EL-GALEN, where the latter one is obtained by removing functionality and inverses from the Full-GALEN ontology.[10] Gazetteer, Biomodels, and Cell Cycle v2.01 are large but mainly deterministic ontologies from the NCBO BioPortal, NCI v06.12d and NCI v12.11d are different versions of the NCI-Thesaurus ontology from the NCIt archive, SCT-SEP denotes the SNOMED CT anatomical model ontology,[11] and OBI represents a recent version of the OBI ontology.[12] Please note that some of these ontologies are also part of the test corpus.

The evaluation was carried out on a Dell PowerEdge R420 server running with two Intel Xeon E5-2440 hexa core processors at 2.4 GHz with Hyper-Threading and 48 GB RAM under a 64bit Ubuntu 12.04.2 LTS. Our evaluation focuses on classification, which is a central reasoning task that is supported by many reasoners and, thus, it is ideal for the comparison of results. In principle, we only measured the classification times, i.e., the times spent for parsing and loading ontologies as well as writing classi-

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

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

[7] `http://www.cs.ox.ac.uk/isg/ontologies/`; Note, the Oxford ontology library also contains other repositories (e.g., the Gardiner ontology suite), which we ignored in order to avoid too much redundancy.

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

[9] 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.

[10] `http://www.co-ode.org/galen/`

[11] `http://condor-reasoner.googlecode.com/`

[12] `http://obi-ontology.org/`

fication output to files are not included for the presented 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 1,400 s for Konclude, 4,130 s for FaCT++, 4,192 s for MORe, 6,452 s for Pellet, and 7,869 s for HermiT. In addition, we ignored all errors that were reported by the reasoners, i.e., if a reasoner stopped the processing of an ontology (e.g., due to not supported 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 240, HermiT for 260, FaCT++ for 280, and Pellet for 318 ontologies in our corpus. A frequently reported error consisted of different individual axioms for which only one individual was specified, and HermiT completely refused the processing of ontologies with irregular role inclusion axioms (which are, however, only rarely present in our test corpus).

For the ontology repositories, we used the time limit of 5 minutes, whereas we cancelled the classification task for the selected benchmark ontologies 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 reasoner to use only one worker thread, which allows for a comparison independent of the number of CPU cores and facilitates the presentation of the improvements through saturation.

7.1 Optimisation Evaluation

The presented optimisations are integrated in Konclude such that they can be separately activated and deactivated. Hence, we can evaluate and compare the performance improvements for the different optimisations. Please note that the deactivation of optimisations in Konclude can cause disproportionate performance losses since appropriate replacement optimisations, which could compensate the deactivated techniques at least in a slight way, are often not integrated in Konclude. For example, many reasoning systems are using the completely defined concepts optimisation [30] to identify those classes of an ontology for which all subsumption relations can be directly extracted from the ontology axioms and, thus, satisfiability and subsumption tests are not necessary to correctly insert these classes into the class hierarchy. Obviously, 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 the completion graph (as presented in Section 4.1) is activated/deactivated;
- SE (standing for **s**ubsumer **e**xtraction), where the extraction of subsumers from the saturation (as presented in Section 4.2) is activated/deactivated;
- MM (standing for **m**odel **m**erging), where the model merging with the saturation graph (as presented in Section 4.3) is activated/deactivated;
- 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;
- PS (standing for **p**atched **s**aturation), where the patching of the saturation graph with data from fully expanded and clash-free completion graphs (as presented in Section 5.2) is activated/deactivated.

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 8 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 caused by the model merging optimisation (MM), which is mainly due to the large amount of NCI-Thesaurus ontologies in the NCIt archive, where this optimisation significantly reduces the classification effort. In contrast, the optimisations RT and SE only allow for minor improvements for the accumulated classification times. Nevertheless, if all saturation optimisations are activated, then we can again observe a significant performance improvement for all repositories, which shows that there is a further synergy effect from the combination of the different optimisations. Obviously, this is hardly surprising since at least the saturation is shared by all optimisations.

Table 9 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. 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 deterministic ontologies do not have possible subsumers for which the model merging could be applied.

Table 10 shows the performance changes for the separate deactivation of saturation optimisations based on the ALL configuration. The evaluation of optimisations might be more interesting from this perspective, because the saturation of many concepts can

Table 8. Accumulated classification times (in seconds) with separately activated saturation optimisations for the evaluated ontology repositories

Repository	NONE	NONE+RT	NONE+SE	NONE+MM	ALL
Gardiner	531	611	469	535	559
NCBO BioPortal	2,071	1,947	971	2,156	793
NCIt	28,639	28,538	28,276	3,223	2,457
OBO Foundry	879	821	979	1,078	649
Oxford	6,623	5,006	6,012	6,510	2,743
TONES	1,756	1,456	1,413	494	337
Google Crawl	465	428	448	467	138
OntoCrawler	26	25	24	25	22
OntoJCrawl	1,417	923	715	1,427	548
Swoogle Crawl	2,501	2,502	2,493	1,402	1,343
ALL	44,910	42,256	41,800	17,317	9,589

Table 9. Accumulated classification times (in seconds) with separately activated saturation optimisations for evaluated benchmark ontologies

Ontology	NONE	NONE+RT	NONE+SE	NONE+MM	ALL
Gazetteer	34.3	36.1	17.0	36.7	16.8
EL-GALEN	73.2	5.8	1.5	76.3	1.5
Biomodels	127.2	56.5	34.5	127.1	34.4
Cell Cycle v2.01	≥ 900.0	≥ 900.0	46.3	≥ 900.0	47.2
NCI v06.12d	≥ 900.0	≥ 900.0	≥ 900.0	23.2	19.6
NCI v12.11d	15.3	15.1	9.9	16.1	8.9
SCT-SEP	≥ 900.0	494.1	218.5	525.7	181.2
OBI	0.9	0.8	0.7	0.9	0.6

easily require a lot of reasoning time and, from this perspective, the overhead of the saturation is not only associated with the separately activated optimisation. Furthermore, this also allows for evaluating the effects of the saturation improvements ES and PS, which is only useful in combination with other saturation optimisations. Table 10 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.

Especially the deactivation of the PS optimisation does often not result in significant performance losses, which can be traced back to several reasons. On the one hand, Konclude integrates several caching techniques which often realise similar tasks as the patching of the saturation graph. On the other hand, the patching of the saturation graph only improves reasoning over time, i.e., critical nodes can only be solved if enough patches are applied. As a consequence, the PS optimisation is usually more important for subsequent reasoning tasks such as realisation. Furthermore, the implementation currently only extracts patches for root nodes from consistency and satisfiability tests, which could be further extended to other nodes if certain conditions are satisfied for

Table 10. Accumulated classification times (in seconds) with separately deactivated saturation optimisations for the evaluated ontology repositories

Repository	ALL	ALL−RT	ALL−SE	ALL−MM	ALL−ES	ALL−PS
Gardiner	559	706	550	505	558	722
NCBO BioPortal	793	992	1,814	797	988	751
NCIt	2,457	2,551	3,019	28,158	2,496	2,451
OBO Foundry	649	793	843	648	741	700
Oxford	2,743	5,899	4,093	2,876	3,429	2,775
TONES	337	251	323	1,457	321	324
Google Crawl	138	488	229	161	363	172
OntoCrawler	22	23	25	21	23	23
OntoJCrawl	548	707	890	406	517	448
Swoogle Crawl	1,343	1,372	1,206	2,428	1,248	1,215
ALL	9,589	13,782	12,993	37,457	10,684	9,583

these nodes. In particular, other caching techniques in Konclude often also consider other nodes, whereby many node labels are already cached if they are identified as resolved in the saturation graph due to patches. Nevertheless, the PS optimisation is often more important for ontologies that are mainly in the OWL 2 EL profile and intensively use nominals since other caching techniques can often not be used in such cases. However, as of now, such ontologies are very rare, especially also due to the fact that they could not be processed by existing reasoners.

The deactivation of MM also causes performance improvements (e.g., for OntoJCrawl and Gardiner), which indicates that further optimisation is possible. For example, one could learn statistics about the success of model merging with certain nodes in the saturation graph and automatically skip a merging test if there is a high "likelihood" that it will fail. Apart from this, the model merging with the saturation graph (MM) again seems to be the most important optimisation due to the NCIt archive.

The performance changes for the separate deactivation of saturation optimisations for the evaluated benchmark ontologies is depicted in Table 11. Again, it can be observed that often only specific optimisations are important for the ontologies. For example, only the deactivation of the SE optimisations produces significant performance losses for the Biomodels ontology.

7.2 Evaluation of Saturation Effort

Clearly, the saturation of many concepts also requires some effort. In the worst case, the saturated concepts are not helpful in any way and, then, the time and memory that were investigated in the saturation are completely wasted. However, as shown in Section 7.1, the combination of the saturation with the presented optimisations significantly improves the overall performance of the reasoning system for many real-world ontologies. Nevertheless, we have evaluated how much time is spent on the saturation in comparison with other processing steps, which can be used as indication where further optimisation is possible. Table 12 shows the percentage of the overall reported processing time that is spent in the different processing stages for the classification tests by the

Table 11. Accumulated classification times (in seconds) with separately deactivated saturation optimisations for selected benchmark ontologies

Ontology	ALL	ALL−RT	ALL−SE	ALL−MM	ALL−ES	ALL−PS
Gazetteer	16.8	16.6	37.2	17.0	17.4	17.4
EL-GALEN	1.5	1.5	5.6	1.5	1.5	1.5
Biomodels	34.4	34.8	53.4	34.4	34.5	34.6
Cell Cycle v2.01	47.2	47.9	53.4	48.3	47.9	45.2
NCI v06.12d	19.6	20.2	21.5	≥ 900.0	19.9	20.0
NCI v12.11d	8.9	9.6	14.3	9.2	8.8	9.2
SCT-SEP	181.2	216.7	500.6	180.4	179.7	179.7
OBI	0.6	0.7	0.8	0.6	0.6	0.6

Table 12. Distribution of processing times in regard to different processing stages (in %)

Repository	Building	Preprocessing	Saturation	Consistency	Classification
Gardiner	1.6	4.6	6.1	0.0	87.7
NCBO BioPortal	8.3	28.8	40.7	1.4	20.8
NCIt	5.3	8.3	13.3	0.0	73.1
OBO Foundry	37.2	10.2	7.8	13.6	31.3
Oxford	7.7	7.0	29.8	11.7	43.8
TONES	1.8	2.2	20.5	0.3	75.1
Google Crawl	8.4	3.6	8.6	3.0	76.3
OntoCrawler	46.4	9.5	5.6	22.7	15.8
OntoJCrawl	12.4	8.0	6.4	5.8	67.5
Swoogle Crawl	3.6	1.2	21.1	57.8	16.3
ALL	8.4	8.4	20.5	12.3	50.4

ALL version of Konclude. Unsurprisingly, the classification requires with 50 % significantly more than any other processing stage for all repositories. However, 20.5 % of the overall processing time is spent for the saturation, which is also a significant amount, but already includes the time that is required for the detection of the saturation status.

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 already available, we evaluated MORe with our test corpus and, in the following, we compare the results to our approach. We used MORe in combination with ELK 0.4.1 [18] 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 13 shows the accumulated classification times 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) in seconds for the different repositories. Furthermore, the improvement from the version

Table 13. Improvements through saturation between the approaches in Konclude and MORe for the accumulated classification times of the evaluated ontology repositories (in seconds and %)

Repository	NONE [s]	ALL [s]	↓ [%]	HermiT [s]	MORe [s]	↓ [%]
Gardiner	531	559	−5.2	1,714	1,590	7.2
NCBO BioPortal	2,071	793	61.7	5,918	3,932	33.6
NCIt	28,639	2,457	91.4	26,435	26,600	−0.6
OBO Foundry	879	649	26.2	5,587	4,627	17.2
Oxford	6,623	2,743	58.6	12,551	8,652	31.1
TONES	1,756	337	80.8	2,609	2,174	16.7
Google Crawl	465	138	70.3	2,147	1,933	10.0
OntoCrawler	26	22	14.7	1,722	876	49.1
OntoJCrawl	1,417	548	61.4	8,253	4,543	45.0
Swoogle Crawl	2,501	1,343	46.3	5,051	4,102	18.8
ALL	44,910	9,589	78.6	72,659	59,028	18.8

Table 14. Improvements through saturation between the approaches in Konclude and MORe for the classification times of selected benchmark ontologies (in seconds and %)

Ontology	NONE [s]	ALL [s]	↓ [%]	HermiT [s]	MORe [s]	↓ [%]
Gazetteer	34.3	16.8	51.2	≥ 900.0	18.2	≥ 98.0
EL-GALEN	73.2	1.5	98.0	≥ 900.0	2.6	≥ 99.7
Biomodels	127.2	34.4	73.0	788.8	648.8	17.7
Cell Cycle v2.01	≥ 900.0	47.2	≥ 94.8	≥ 900.0	≥ 900.0	–
NCI v06.12d	≥ 900.0	19.6	≥ 97.8	211.9	208.0	1.9
NCI v12.11d	15.3	8.9	41.8	92.7	83.3	10.1
SCT-SEP	≥ 900.0	181.2	≥ 79.9	≥ 900.0	≥ 900.0	–
OBI	0.9	0.6	33.3	32.5	2.3	93.0

NONE to the version ALL is given in percent (in column 4 of Table 13). For example, by using all presented saturation optimisations, the accumulated reasoning time for all repositories is reduced by 78.6 % for Konclude. On the right-hand side of Table 13, 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 7,869 s for HermiT and 4,192 s for MORe, where the difference of 3,677 s can be explained by the additional preprocessing that is already performed in HermiT's loading stage. In order to get a fair comparison, we added the preprocessing time from 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.

Table 13 reveals that MORe can significantly improve the reasoning time of HermiT for almost all repositories. In particular, MORe saves 49.1 % 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 129 and MORe for 103 ontologies, respectively). In contrast, Konclude integrates a more sophisticated in-

teraction 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 ALL version of Konclude reached the time limit only for 10 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.

Table 14 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 performance of the version NONE of Konclude is not already good.

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 15. Note, Table 15 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 this allows Konclude to outperform FaCT++ for the NCIt archive.

Analogously, Table 16 shows the comparison of the classification times between all evaluated reasoners for the selected benchmark ontologies in seconds. Again, with the activated 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. In comparison with the remaining reasoners, the results of MORe are also very good. In particular, MORe only timed out for 2 ontologies. Hence, an assistance through saturation seems to pay off.

8 Conclusions and Future Work

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.

[13] FaCT++ 1.6.2 crashed for the classification of Biomodels after 2.7 seconds.

Table 15. Comparison of accumulated classification times between state-of-the-art reasoners (in seconds) for the evaluated ontology repositories

Repository	FaCT++	HermiT	Konclude	MORe	Pellet
Gardiner	3,122	1,675	559	1,590	4,004
NCBO BioPortal	5,230	5,702	793	3,932	9,226
NCIt	3,892	25,203	2,457	26,600	17,647
OBO Foundry	7,168	6,075	649	4,627	12,031
Oxford	23,581	12,155	2,743	8,652	27,450
TONES	1,678	2,241	337	2,174	2,391
Google Crawl	1,946	2,100	138	1,933	7,733
OntoCrawler	1,077	1,694	22	876	9,894
OntoJCrawl	14,125	6,907	548	4,543	30,396
Swoogle Crawl	3,281	4,558	1,343	4,102	9,026
ALL	65,100	68,310	9,589	59,028	129,798

Table 16. Comparison of accumulated classification times between state-of-the-art reasoners (in seconds) for selected benchmark ontologies

Ontology	FaCT++	HermiT	Konclude	MORe	Pellet
Gazetteer	≥ 900.0	≥ 900.0	16.8	18.2	480.2
EL-GALEN	≥ 900.0	≥ 900.0	1.5	2.6	135.1
Biomodels	2.7^{13}	788.8	34.4	648.8	≥ 900.0
Cell Cycle v2.01	≥ 900.0	≥ 900.0	47.2	≥ 900.0	≥ 900.0
NCI v06.12d	13.9	206.1	19.6	208.0	69.6
NCI v12.11d	57.8	78.8	8.9	83.3	306.9
SCT-SEP	≥ 900.0	≥ 900.0	181.2	≥ 900.0	≥ 900.0
OBI	≥ 900.0	31.5	0.6	2.3	≥ 900.0

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 by often more than one order of magnitude.

There are also several possibilities to enhance our approach in future works. In particular, the saturation can be extended to support features of more expressive Description Logics in more detail. As discussed, more precise saturation extensions for nominals are possible and, as soon as corresponding ontologies are available, their integration and evaluation could be very interesting. Moreover, extensions to support datatypes should be more or less straightforward. Due to the very good pay-as-you-go behaviour, our approach promises a very good handling of ontologies even if datatypes are used beyond the rigid restrictions of less expressive Description Logics. As already considered for other reasoners, we could also generate an upper bound approximation with the saturation (e.g., for disjunction, we could consider the addition of all disjuncts at the same time), and then we could identify which new consequences can be inferred

compared to the ordinary saturation. This could be used to quickly decide the satisfiability of sets of concepts and, thus, to further assist and guide the tableau algorithm.

Since some saturation optimisations already overlap with other optimisation techniques integrated in Konclude, we can examine whether the deactivation, reconfiguration, or specialisation of these techniques improves the reasoning performance. In particular, other caching techniques can possibly be reduced since the saturation-based optimisations often realise the same tasks and are usually faster.

Acknowledgements

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

References

1. Armas Romero, A., Cuenca Grau, B., Horrocks, I.: MORe: Modular combination of OWL reasoners for ontology classification. In: Proc. 11th Int. Semantic Web Conf. (ISWC'12). LNCS, vol. 7649, pp. 1–16. Springer (2012)
2. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proc. 19th Int. Joint Conf. on Artificial Intelligence (IJCAI'05). pp. 364–369. Professional Book Center (2005)
3. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, second edn. (2007)
4. Baader, F., Hollunder, B., Nebel, B., Profitlich, H.J., Franconi, E.: An empirical analysis of optimization techniques for terminological representation systems. J. of Applied Intelligence 4(2), 109–132 (1994)
5. Gardiner, T., Horrocks, I., Tsarkov, D.: Automated benchmarking of description logic reasoners. In: Proc. 19th Int. Workshop on Description Logics (DL'06). vol. 198. CEUR (2006)
6. Glimm, B., Horrocks, I., Motik, B., Shearer, R., Stoilos, G.: A novel approach to ontology classification. J. of Web Semantics 14, 84–101 (2012)
7. Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: Hermit: An OWL 2 reasoner. J. of Automated Reasoning pp. 1–25 (2014)
8. Haarslev, V., Möller, R., Turhan, A.Y.: Exploiting pseudo models for tbox and abox reasoning in expressive description logics. In: Proc. 1st Int. Joint Conf. on Automated Reasoning (IJCAR'01). LNCS, vol. 2083, pp. 61–75. Springer (2001)
9. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: Proc. 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'06). pp. 57–67. AAAI Press (2006)
10. Horrocks, I., Sattler, U.: A description logic with transitive and inverse roles and role hierarchies. J. of Logic and Computation 9(3), 385–410 (1999)
11. Horrocks, I., Sattler, U.: Optimised reasoning for $\mathcal{SHIQ}$. In: Proc. 15th European Conf. on Artificial Intelligence (ECAI'02). pp. 277–281. IOS Press (2001)
12. Horrocks, I., Sattler, U., Tobies, S.: Reasoning with individuals for the description logic $\mathcal{SHIQ}$. In: Proc. 17th Int. Conf. on Automated Deduction (CADE'00). Lecture Notes in Computer Science, vol. 1831, pp. 482–496. Springer (2000)
13. Horrocks, I., Tobies, S.: Reasoning with axioms: Theory and practice. In: Proc. 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'00). pp. 285–296. Morgan Kaufmann (2000)

14. Hudek, A.K., Weddell, G.E.: Binary absorption in tableaux-based reasoning for description logics. In: Proc. 19th Int. Workshop on Description Logics (DL'06). vol. 189. CEUR (2006)
15. Kazakov, Y.: $\mathcal{RIQ}$ and $\mathcal{SROIQ}$ are harder than $\mathcal{SHOIQ}$. In: Proc. 11th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'08). pp. 274–284. AAAI Press (2008)
16. Kazakov, Y.: Consequence-driven reasoning for Horn-$\mathcal{SHIQ}$ ontologies. In: Proc. 21st Int. Conf. on Artificial Intelligence (IJCAI'09). pp. 2040–2045. IJCAI (2009)
17. Kazakov, Y., Krötzsch, M., Simančík, F.: 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 (2012)
18. Kazakov, Y., Krötzsch, M., Simančík, F.: The incredible ELK. J. of Automated Reasoning 53, 1–61 (2014)
19. Matentzoglu, N., Bail, S., Parsia, B.: A corpus of OWL DL ontologies. In: Proc. 26th Int. Workshop on Description Logics (DL'13). vol. 1014. CEUR (2013)
20. Motik, B., Horrocks, I.: Individual reuse in description logic reasoning. In: Proc. 4th Int. Joint Conf. on Automated Reasoning (IJCAR'08). pp. 242–258. No. 5195 in LNCS, Springer (2008)
21. Ortiz, M., Rudolph, S., Simkus, M.: 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 (2010)
22. Simančík, F., Kazakov, Y., Horrocks, I.: Consequence-based reasoning beyond horn ontologies. In: Proc. 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI'11). pp. 1093–1098. IJCAI/AAAI (2011)
23. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. J. of Web Semantics 5(2), 51–53 (2007)
24. 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.: The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. Nature Biotechnology 25, 1251–1255 (2007)
25. Song, W., Spencer, B., Du, W.: 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 (2012)
26. Steigmiller, A., Glimm, B., Liebig, T.: Nominal schema absorption. In: Proc. 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI'13). pp. 1104–1110. AAAI Press (2013)
27. Steigmiller, A., Glimm, B., Liebig, T.: Optimised absorption for expressive description logics. In: Proc. 27th Int. Workshop on Description Logics (DL'14) (2014), accepted
28. Steigmiller, A., Liebig, T., Glimm, B.: Extended caching, backjumping and merging for expressive description logics. In: Proc. 6th Int. Joint Conf. on Automated Reasoning (IJCAR'12). LNCS, vol. 7364, pp. 514–529. Springer (2012)
29. Steigmiller, A., Liebig, T., Glimm, B.: Konclude: system description. J. of Web Semantics (2014), accepted
30. Tsarkov, D., Horrocks, I.: Optimised classification for taxonomic knowledge bases. In: Proc. 18th Int. Workshop on Description Logics (DL'05). vol. 147. CEUR (2005)
31. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In: Proc. 3rd Int. Joint Conf. on Automated Reasoning (IJCAR'06). LNCS, vol. 4130, pp. 292–297. Springer (2006)
32. Tsarkov, D., Horrocks, I., Patel-Schneider, P.F.: Optimizing terminological reasoning for expressive description logics. J. of Automated Reasoning 39, 277–316 (2007)
33. W3C OWL Working Group: OWL 2 Web Ontology Language: Document Overview. W3C Recommendation (27 October 2009), available at `http://www.w3.org/TR/owl2-overview/`

Liste der bisher erschienenen Ulmer Informatik-Berichte

Einige davon sind per FTP von `ftp.informatik.uni-ulm.de` erhältlich
Die mit * markierten Berichte sind vergriffen

List of technical reports published by the University of Ulm

Some of them are available by FTP from `ftp.informatik.uni-ulm.de`
Reports marked with * are out of print

91-01 *Ker-I Ko, P. Orponen, U. Schöning, O. Watanabe*
Instance Complexity

*91-02** *K. Gladitz, H. Fassbender, H. Vogler*
Compiler-Based Implementation of Syntax-Directed Functional Programming

*91-03** *Alfons Geser*
Relative Termination

*91-04** *J. Köbler, U. Schöning, J. Toran*
Graph Isomorphism is low for PP

91-05 *Johannes Köbler, Thomas Thierauf*
Complexity Restricted Advice Functions

*91-06** *Uwe Schöning*
Recent Highlights in Structural Complexity Theory

*91-07** *F. Green, J. Köbler, J. Toran*
The Power of Middle Bit

*91-08** *V.Arvind, Y. Han, L. Hamachandra, J. Köbler, A. Lozano, M. Mundhenk, A. Ogiwara, U. Schöning, R. Silvestri, T. Thierauf*
Reductions for Sets of Low Information Content

*92-01** *Vikraman Arvind, Johannes Köbler, Martin Mundhenk*
On Bounded Truth-Table and Conjunctive Reductions to Sparse and Tally Sets

*92-02** *Thomas Noll, Heiko Vogler*
Top-down Parsing with Simulataneous Evaluation of Noncircular Attribute Grammars

92-03 *Fakultät für Informatik*
17. Workshop über Komplexitätstheorie, effiziente Algorithmen und Datenstrukturen

*92-04** *V. Arvind, J. Köbler, M. Mundhenk*
Lowness and the Complexity of Sparse and Tally Descriptions

*92-05** *Johannes Köbler*
Locating P/poly Optimally in the Extended Low Hierarchy

*92-06** *Armin Kühnemann, Heiko Vogler*
Synthesized and inherited functions -a new computational model for syntax-directed semantics

*92-07** *Heinz Fassbender, Heiko Vogler*
A Universal Unification Algorithm Based on Unification-Driven Leftmost Outermost Narrowing

*92-08** *Uwe Schöning*
On Random Reductions from Sparse Sets to Tally Sets

*92-09** *Hermann von Hasseln, Laura Martignon*
Consistency in Stochastic Network

92-10 *Michael Schmitt*
A Slightly Improved Upper Bound on the Size of Weights Sufficient to Represent Any Linearly Separable Boolean Function

92-11 *Johannes Köbler, Seinosuke Toda*
On the Power of Generalized MOD-Classes

92-12 *V. Arvind, J. Köbler, M. Mundhenk*
Reliable Reductions, High Sets and Low Sets

92-13 *Alfons Geser*
On a monotonic semantic path ordering

*92-14** *Joost Engelfriet, Heiko Vogler*
The Translation Power of Top-Down Tree-To-Graph Transducers

93-01 *Alfred Lupper, Konrad Froitzheim*
AppleTalk Link Access Protocol basierend auf dem Abstract Personal Communications Manager

93-02 *M.H. Scholl, C. Laasch, C. Rich, H.-J. Schek, M. Tresch*
The COCOON Object Model

93-03 *Thomas Thierauf, Seinosuke Toda, Osamu Watanabe*
On Sets Bounded Truth-Table Reducible to P-selective Sets

93-04 *Jin-Yi Cai, Frederic Green, Thomas Thierauf*
On the Correlation of Symmetric Functions

93-05 *K.Kuhn, M.Reichert, M. Nathe, T. Beuter, C. Heinlein, P. Dadam*
A Conceptual Approach to an Open Hospital Information System

93-06 *Klaus Gaßner*
Rechnerunterstützung für die konzeptuelle Modellierung

93-07 *Ullrich Keßler, Peter Dadam*
Towards Customizable, Flexible Storage Structures for Complex Objects

94-01 *Michael Schmitt*
On the Complexity of Consistency Problems for Neurons with Binary Weights

94-02 *Armin Kühnemann, Heiko Vogler*
A Pumping Lemma for Output Languages of Attributed Tree Transducers

94-03 *Harry Buhrman, Jim Kadin, Thomas Thierauf*
On Functions Computable with Nonadaptive Queries to NP

94-04 *Heinz Faßbender, Heiko Vogler, Andrea Wedel*
Implementation of a Deterministic Partial E-Unification Algorithm for Macro Tree Transducers

94-05 *V. Arvind, J. Köbler, R. Schuler*
On Helping and Interactive Proof Systems

94-06 *Christian Kalus, Peter Dadam*
Incorporating record subtyping into a relational data model

94-07 *Markus Tresch, Marc H. Scholl*
A Classification of Multi-Database Languages

94-08 *Friedrich von Henke, Harald Rueß*
Arbeitstreffen Typtheorie: Zusammenfassung der Beiträge

94-09 *F.W. von Henke, A. Dold, H. Rueß, D. Schwier, M. Strecker*
Construction and Deduction Methods for the Formal Development of Software

94-10 *Axel Dold*
Formalisierung schematischer Algorithmen

94-11 *Johannes Köbler, Osamu Watanabe*
New Collapse Consequences of NP Having Small Circuits

94-12 *Rainer Schuler*
On Average Polynomial Time

94-13 *Rainer Schuler, Osamu Watanabe*
Towards Average-Case Complexity Analysis of NP Optimization Problems

94-14 *Wolfram Schulte, Ton Vullinghs*
Linking Reactive Software to the X-Window System

94-15 *Alfred Lupper*
Namensverwaltung und Adressierung in Distributed Shared Memory-Systemen

94-16 *Robert Regn*
Verteilte Unix-Betriebssysteme

94-17 *Helmuth Partsch*
Again on Recognition and Parsing of Context-Free Grammars:
Two Exercises in Transformational Programming

94-18 *Helmuth Partsch*
Transformational Development of Data-Parallel Algorithms: an Example

95-01 *Oleg Verbitsky*
On the Largest Common Subgraph Problem

95-02 *Uwe Schöning*
Complexity of Presburger Arithmetic with Fixed Quantifier Dimension

95-03 *Harry Buhrman,Thomas Thierauf*
The Complexity of Generating and Checking Proofs of Membership

95-04 *Rainer Schuler, Tomoyuki Yamakami*
Structural Average Case Complexity

95-05 *Klaus Achatz, Wolfram Schulte*
Architecture Indepentent Massive Parallelization of Divide-And-Conquer Algorithms

95-06 *Christoph Karg, Rainer Schuler*
Structure in Average Case Complexity

95-07 *P. Dadam, K. Kuhn, M. Reichert, T. Beuter, M. Nathe*
ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen

95-08 *Jürgen Kehrer, Peter Schulthess*
Aufbereitung von gescannten Röntgenbildern zur filmlosen Diagnostik

95-09 *Hans-Jörg Burtschick, Wolfgang Lindner*
On Sets Turing Reducible to P-Selective Sets

95-10 *Boris Hartmann*
Berücksichtigung lokaler Randbedingung bei globaler Zieloptimierung mit neuronalen Netzen am Beispiel Truck Backer-Upper

95-11 *Thomas Beuter, Peter Dadam*:
Prinzipien der Replikationskontrolle in verteilten Systemen

95-12 *Klaus Achatz, Wolfram Schulte*
Massive Parallelization of Divide-and-Conquer Algorithms over Powerlists

95-13 *Andrea Mößle, Heiko Vogler*
Efficient Call-by-value Evaluation Strategy of Primitive Recursive Program Schemes

95-14 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
A Generic Specification for Verifying Peephole Optimizations

96-01 *Ercüment Canver, Jan-Tecker Gayen, Adam Moik*
Formale Entwicklung der Steuerungssoftware für eine elektrisch ortsbediente Weiche mit VSE

96-02 *Bernhard Nebel*
Solving Hard Qualitative Temporal Reasoning Problems: Evaluating the Efficiency of Using the ORD-Horn Class

96-03 *Ton Vullinghs, Wolfram Schulte, Thilo Schwinn*
An Introduction to TkGofer

96-04 *Thomas Beuter, Peter Dadam*
Anwendungsspezifische Anforderungen an Workflow-Mangement-Systeme am Beispiel der Domäne Concurrent-Engineering

96-05 *Gerhard Schellhorn, Wolfgang Ahrendt*
Verification of a Prolog Compiler - First Steps with KIV

96-06 *Manindra Agrawal, Thomas Thierauf*
Satisfiability Problems

96-07 *Vikraman Arvind, Jacobo Torán*
A nonadaptive NC Checker for Permutation Group Intersection

96-08 *David Cyrluk, Oliver Möller, Harald Rueß*
An Efficient Decision Procedure for a Theory of Fix-Sized Bitvectors with Composition and Extraction

96-09 *Bernd Biechele, Dietmar Ernst, Frank Houdek, Joachim Schmid, Wolfram Schulte*
Erfahrungen bei der Modellierung eingebetteter Systeme mit verschiedenen SA/RT–Ansätzen

96-10 *Falk Bartels, Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Formalizing Fixed-Point Theory in PVS

96-11 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Mechanized Semantics of Simple Imperative Programming Constructs

96-12 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Generic Compilation Schemes for Simple Programming Constructs

96-13 *Klaus Achatz, Helmuth Partsch*
From Descriptive Specifications to Operational ones: A Powerful Transformation Rule, its Applications and Variants

97-01 *Jochen Messner*
Pattern Matching in Trace Monoids

97-02 *Wolfgang Lindner, Rainer Schuler*
A Small Span Theorem within P

97-03 *Thomas Bauer, Peter Dadam*
A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration

97-04 *Christian Heinlein, Peter Dadam*
Interaction Expressions - A Powerful Formalism for Describing Inter-Workflow Dependencies

97-05 *Vikraman Arvind, Johannes Köbler*
On Pseudorandomness and Resource-Bounded Measure

97-06 *Gerhard Partsch*
Punkt-zu-Punkt- und Mehrpunkt-basierende LAN-Integrationsstrategien für den digitalen Mobilfunkstandard DECT

97-07 *Manfred Reichert, Peter Dadam*
$ADEPT_{flex}$ - Supporting Dynamic Changes of Workflows Without Loosing Control

97-08 *Hans Braxmeier, Dietmar Ernst, Andrea Mößle, Heiko Vogler*
The Project NoName - A functional programming language with its development environment

97-09 *Christian Heinlein*
Grundlagen von Interaktionsausdrücken

97-10 *Christian Heinlein*
Graphische Repräsentation von Interaktionsausdrücken

97-11 *Christian Heinlein*
Sprachtheoretische Semantik von Interaktionsausdrücken

97-12 *Gerhard Schellhorn, Wolfgang Reif*
Proving Properties of Finite Enumerations: A Problem Set for Automated Theorem Provers

97-13 *Dietmar Ernst, Frank Houdek, Wolfram Schulte, Thilo Schwinn*
Experimenteller Vergleich statischer und dynamischer Softwareprüfung für eingebettete Systeme

97-14 *Wolfgang Reif, Gerhard Schellhorn*
Theorem Proving in Large Theories

97-15 *Thomas Wennekers*
Asymptotik rekurrenter neuronaler Netze mit zufälligen Kopplungen

97-16 *Peter Dadam, Klaus Kuhn, Manfred Reichert*
Clinical Workflows - The Killer Application for Process-oriented Information Systems?

97-17 *Mohammad Ali Livani, Jörg Kaiser*
EDF Consensus on CAN Bus Access in Dynamic Real-Time Applications

97-18 *Johannes Köbler,Rainer Schuler*
Using Efficient Average-Case Algorithms to Collapse Worst-Case Complexity Classes

98-01 *Daniela Damm, Lutz Claes, Friedrich W. von Henke, Alexander Seitz, Adelinde Uhrmacher, Steffen Wolf*
Ein fallbasiertes System für die Interpretation von Literatur zur Knochenheilung

98-02 *Thomas Bauer, Peter Dadam*
Architekturen für skalierbare Workflow-Management-Systeme - Klassifikation und Analyse

98-03 *Marko Luther, Martin Strecker*
A guided tour through *Typelab*

98-04 *Heiko Neumann, Luiz Pessoa*
Visual Filling-in and Surface Property Reconstruction

98-05 *Ercüment Canver*
Formal Verification of a Coordinated Atomic Action Based Design

98-06 *Andreas Küchler*
On the Correspondence between Neural Folding Architectures and Tree Automata

98-07 *Heiko Neumann, Thorsten Hansen, Luiz Pessoa*
Interaction of ON and OFF Pathways for Visual Contrast Measurement

98-08 *Thomas Wennekers*
Synfire Graphs: From Spike Patterns to Automata of Spiking Neurons

98-09 *Thomas Bauer, Peter Dadam*
Variable Migration von Workflows in *ADEPT*

98-10 *Heiko Neumann, Wolfgang Sepp*
Recurrent V1 – V2 Interaction in Early Visual Boundary Processing

98-11 *Frank Houdek, Dietmar Ernst, Thilo Schwinn*
Prüfen von C–Code und Statmate/Matlab–Spezifikationen: Ein Experiment

98-12 *Gerhard Schellhorn*
Proving Properties of Directed Graphs: A Problem Set for Automated Theorem Provers

98-13 *Gerhard Schellhorn, Wolfgang Reif*
Theorems from Compiler Verification: A Problem Set for Automated Theorem Provers

98-14 *Mohammad Ali Livani*
SHARE: A Transparent Mechanism for Reliable Broadcast Delivery in CAN

98-15 *Mohammad Ali Livani, Jörg Kaiser*
Predictable Atomic Multicast in the Controller Area Network (CAN)

99-01 *Susanne Boll, Wolfgang Klas, Utz Westermann*
A Comparison of Multimedia Document Models Concerning Advanced Requirements

99-02 *Thomas Bauer, Peter Dadam*
Verteilungsmodelle für Workflow-Management-Systeme - Klassifikation und Simulation

99-03 *Uwe Schöning*
On the Complexity of Constraint Satisfaction

99-04 *Ercument Canver*
Model-Checking zur Analyse von Message Sequence Charts über Statecharts

99-05 *Johannes Köbler, Wolfgang Lindner, Rainer Schuler*
Derandomizing RP if Boolean Circuits are not Learnable

99-06 *Utz Westermann, Wolfgang Klas*
Architecture of a DataBlade Module for the Integrated Management of Multimedia Assets

99-07 *Peter Dadam, Manfred Reichert*
Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications. Paderborn, Germany, October 6, 1999, GI–Workshop Proceedings, Informatik ’99

99-08 *Vikraman Arvind, Johannes Köbler*
Graph Isomorphism is Low for ZPP^{NP} and other Lowness results

99-09 *Thomas Bauer, Peter Dadam*
Efficient Distributed Workflow Management Based on Variable Server Assignments

2000-02 *Thomas Bauer, Peter Dadam*
Variable Serverzuordnungen und komplexe Bearbeiterzuordnungen im Workflow-Management-System ADEPT

2000-03 *Gregory Baratoff, Christian Toepfer, Heiko Neumann*
Combined space-variant maps for optical flow based navigation

2000-04 *Wolfgang Gehring*
Ein Rahmenwerk zur Einführung von Leistungspunktsystemen

2000-05 *Susanne Boll, Christian Heinlein, Wolfgang Klas, Jochen Wandel*
Intelligent Prefetching and Buffering for Interactive Streaming of MPEG Videos

2000-06 *Wolfgang Reif, Gerhard Schellhorn, Andreas Thums*
Fehlersuche in Formalen Spezifikationen

2000-07 *Gerhard Schellhorn, Wolfgang Reif (eds.)*
FM-Tools 2000: The 4th Workshop on Tools for System Design and Verification

2000-08 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Effiziente Durchführung von Prozessmigrationen in verteilten Workflow-Management-Systemen

2000-09 *Thomas Bauer, Peter Dadam*
Vermeidung von Überlastsituationen durch Replikation von Workflow-Servern in ADEPT

2000-10 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Adaptives und verteiltes Workflow-Management

2000-11 *Christian Heinlein*
Workflow and Process Synchronization with Interaction Expressions and Graphs

2001-01 *Hubert Hug, Rainer Schuler*
DNA-based parallel computation of simple arithmetic

2001-02 *Friedhelm Schwenker, Hans A. Kestler, Günther Palm*
3-D Visual Object Classification with Hierarchical Radial Basis Function Networks

2001-03 *Hans A. Kestler, Friedhelm Schwenker, Günther Palm*
RBF network classification of ECGs as a potential marker for sudden cardiac death

2001-04 *Christian Dietrich, Friedhelm Schwenker, Klaus Riede, Günther Palm*
Classification of Bioacoustic Time Series Utilizing Pulse Detection, Time and Frequency Features and Data Fusion

2002-01 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Effiziente Verträglichkeitsprüfung und automatische Migration von Workflow-Instanzen bei der Evolution von Workflow-Schemata

2002-02 *Walter Guttmann*
Deriving an Applicative Heapsort Algorithm

2002-03 *Axel Dold, Friedrich W. von Henke, Vincent Vialard, Wolfgang Goerigk*
A Mechanically Verified Compiling Specification for a Realistic Compiler

2003-01 *Manfred Reichert, Stefanie Rinderle, Peter Dadam*
A Formal Framework for Workflow Type and Instance Changes Under Correctness Checks

2003-02 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Supporting Workflow Schema Evolution By Efficient Compliance Checks

2003-03 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values

2003-04 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
On Dealing With Semantically Conflicting Business Process Changes.

2003-05 *Christian Heinlein*
Dynamic Class Methods in Java

2003-06 *Christian Heinlein*
Vertical, Horizontal, and Behavioural Extensibility of Software Systems

2003-07 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values
(Corrected Version)

2003-08 *Changling Liu, Jörg Kaiser*
Survey of Mobile Ad Hoc Network Routing Protocols)

2004-01 *Thom Frühwirth, Marc Meister (eds.)*
First Workshop on Constraint Handling Rules

2004-02 *Christian Heinlein*
Concept and Implementation of C+++, an Extension of C++ to Support User-Defined Operator Symbols and Control Structures

2004-03 *Susanne Biundo, Thom Frühwirth, Günther Palm(eds.)*
Poster Proceedings of the 27th Annual German Conference on Artificial Intelligence

2005-01 *Armin Wolf, Thom Frühwirth, Marc Meister (eds.)*
19th Workshop on (Constraint) Logic Programming

2005-02 *Wolfgang Lindner (Hg.), Universität Ulm , Christopher Wolf (Hg.) KU Leuven*
2. Krypto-Tag – Workshop über Kryptographie, Universität Ulm

2005-03 *Walter Guttmann, Markus Maucher*
Constrained Ordering

2006-01 *Stefan Sarstedt*
Model-Driven Development with ACTIVECHARTS, Tutorial

2006-02 *Alexander Raschke, Ramin Tavakoli Kolagari*
Ein experimenteller Vergleich zwischen einer plan-getriebenen und einer leichtgewichtigen Entwicklungsmethode zur Spezifikation von eingebetteten Systemen

2006-03 *Jens Kohlmeyer, Alexander Raschke, Ramin Tavakoli Kolagari*
Eine qualitative Untersuchung zur Produktlinien-Integration über Organisationsgrenzen hinweg

2006-04 *Thorsten Liebig*
Reasoning with OWL - System Support and Insights –

2008-01 *H.A. Kestler, J. Messner, A. Müller, R. Schuler*
On the complexity of intersecting multiple circles for graphical display

2008-02 *Manfred Reichert, Peter Dadam, Martin Jurisch,l Ulrich Kreher, Kevin Göser, Markus Lauer*
Architectural Design of Flexible Process Management Technology

2008-03 *Frank Raiser*
Semi-Automatic Generation of CHR Solvers from Global Constraint Automata

2008-04 *Ramin Tavakoli Kolagari, Alexander Raschke, Matthias Schneiderhan, Ian Alexander*
Entscheidungsdokumentation bei der Entwicklung innovativer Systeme für produktlinien-basierte Entwicklungsprozesse

2008-05 *Markus Kalb, Claudia Dittrich, Peter Dadam*
Support of Relationships Among Moving Objects on Networks

2008-06 *Matthias Frank, Frank Kargl, Burkhard Stiller (Hg.)*
WMAN 2008 – KuVS Fachgespräch über Mobile Ad-hoc Netzwerke

2008-07 *M. Maucher, U. Schöning, H.A. Kestler*
An empirical assessment of local and population based search methods with different degrees of pseudorandomness

2008-08 *Henning Wunderlich*
Covers have structure

2008-09 *Karl-Heinz Niggl, Henning Wunderlich*
Implicit characterization of FPTIME and NC revisited

2008-10 *Henning Wunderlich*
On span-P^{cc} and related classes in structural communication complexity

2008-11 *M. Maucher, U. Schöning, H.A. Kestler*
On the different notions of pseudorandomness

2008-12 *Henning Wunderlich*
On Toda's Theorem in structural communication complexity

2008-13 *Manfred Reichert, Peter Dadam*
Realizing Adaptive Process-aware Information Systems with ADEPT2

2009-01 *Peter Dadam, Manfred Reichert*
The ADEPT Project: A Decade of Research and Development for Robust and Fexible Process Support
Challenges and Achievements

2009-02 *Peter Dadam, Manfred Reichert, Stefanie Rinderle-Ma, Kevin Göser, Ulrich Kreher, Martin Jurisch*
Von ADEPT zur AristaFlow® BPM Suite – Eine Vision wird Realität "Correctness by Construction" und flexible, robuste Ausführung von Unternehmensprozessen

2009-03 *Alena Hallerbach, Thomas Bauer, Manfred Reichert*
Correct Configuration of Process Variants in Provop

2009-04 *Martin Bader*
On Reversal and Transposition Medians

2009-05 *Barbara Weber, Andreas Lanz, Manfred Reichert*
Time Patterns for Process-aware Information Systems: A Pattern-based Analysis

2009-06 *Stefanie Rinderle-Ma, Manfred Reichert*
Adjustment Strategies for Non-Compliant Process Instances

2009-07 *H.A. Kestler, B. Lausen, H. Binder H.-P. Klenk. F. Leisch, M. Schmid*
Statistical Computing 2009 – Abstracts der 41. Arbeitstagung

2009-08 *Ulrich Kreher, Manfred Reichert, Stefanie Rinderle-Ma, Peter Dadam*
Effiziente Repräsentation von Vorlagen- und Instanzdaten in Prozess-Management-Systemen

2009-09 *Dammertz, Holger, Alexander Keller, Hendrik P.A. Lensch*
Progressive Point-Light-Based Global Illumination

2009-10 *Dao Zhou, Christoph Müssel, Ludwig Lausser, Martin Hopfensitz, Michael Kühl, Hans A. Kestler*
Boolean networks for modeling and analysis of gene regulation

2009-11 *J. Hanika, H.P.A. Lensch, A. Keller*
Two-Level Ray Tracing with Recordering for Highly Complex Scenes

2009-12 *Stephan Buchwald, Thomas Bauer, Manfred Reichert*
Durchgängige Modellierung von Geschäftsprozessen durch Einführung eines Abbildungsmodells: Ansätze, Konzepte, Notationen

2010-01 *Hariolf Betz, Frank Raiser, Thom Frühwirth*
A Complete and Terminating Execution Model for Constraint Handling Rules

2010-02 *Ulrich Kreher, Manfred Reichert*
Speichereffiziente Repräsentation instanzspezifischer Änderungen in Prozess-Management-Systemen

2010-03 *Patrick Frey*
Case Study: Engine Control Application

2010-04 *Matthias Lohrmann und Manfred Reichert*
Basic Considerations on Business Process Quality

2010-05 *HA Kestler, H Binder, B Lausen, H-P Klenk, M Schmid, F Leisch (eds):*
Statistical Computing 2010 - Abstracts der 42. Arbeitstagung

2010-06 *Vera Künzle, Barbara Weber, Manfred Reichert*
Object-aware Business Processes: Properties, Requirements, Existing Approaches

2011-01 *Stephan Buchwald, Thomas Bauer, Manfred Reichert*
Flexibilisierung Service-orientierter Architekturen

2011-02 *Johannes Hanika, Holger Dammertz, Hendrik Lensch*
Edge-Optimized À-Trous Wavelets for Local Contrast Enhancement with Robust Denoising

2011-03 *Stefanie Kaiser, Manfred Reichert*
Datenflussvarianten in Prozessmodellen: Szenarien, Herausforderungen, Ansätze

2011-04 *Hans A. Kestler, Harald Binder, Matthias Schmid, Friedrich Leisch, Johann M. Kraus (eds):*
Statistical Computing 2011 - Abstracts der 43. Arbeitstagung

2011-05 *Vera Künzle, Manfred Reichert*
PHILharmonicFlows: Research and Design Methodology

2011-06 *David Knuplesch, Manfred Reichert*
Ensuring Business Process Compliance Along the Process Life Cycle

2011-07 *Marcel Dausend*
Towards a UML Profile on Formal Semantics for Modeling Multimodal Interactive Systems

2011-08 *Dominik Gessenharter*
Model-Driven Software Development with ACTIVECHARTS - A Case Study

2012-01 *Andreas Steigmiller, Thorsten Liebig, Birte Glimm*
Extended Caching, Backjumping and Merging for Expressive Description Logics

2012-02 *Hans A. Kestler, Harald Binder, Matthias Schmid, Johann M. Kraus (eds):*
Statistical Computing 2012 - Abstracts der 44. Arbeitstagung

2012-03 *Felix Schüssel, Frank Honold, Michael Weber*
Influencing Factors on Multimodal Interaction at Selection Tasks

2012-04 *Jens Kolb, Paul Hübner, Manfred Reichert*
Model-Driven User Interface Generation and Adaption in Process-Aware Information Systems

2012-05 *Matthias Lohrmann, Manfred Reichert*
Formalizing Concepts for Efficacy-aware Business Process Modeling

2012-06 *David Knuplesch, Rüdiger Pryss, Manfred Reichert*
A Formal Framework for Data-Aware Process Interaction Models

2012-07 *Clara Ayora, Victoria Torres, Barbara Weber, Manfred Reichert, Vicente Pelechano*
Dealing with Variability in Process-Aware Information Systems: Language Requirements, Features, and Existing Proposals

2013-01 *Frank Kargl*
Abstract Proceedings of the 7th Workshop on Wireless and Mobile Ad-Hoc Networks (WMAN 2013)

2013-02 *Andreas Lanz, Manfred Reichert, Barbara Weber*
A Formal Semantics of Time Patterns for Process-aware Information Systems

2013-03 *Matthias Lohrmann, Manfred Reichert*
Demonstrating the Effectiveness of Process Improvement Patterns with Mining Results

2013-04 *Semra Catalkaya, David Knuplesch, Manfred Reichert*
Bringing More Semantics to XOR-Split Gateways in Business Process Models Based on Decision Rules

2013-05 *David Knuplesch, Manfred Reichert, Linh Thao Ly, Akhil Kumar, Stefanie Rinderle-Ma*
On the Formal Semantics of the Extended Compliance Rule Graph

2013-06 *Andreas Steigmiller, Birte Glimm*
Nominal Schema Absorption

2013-07 *Hans A. Kestler, Matthias Schmid, Florian Schmid, Dr. Markus Maucher, Johann M. Kraus (eds)*
Statistical Computing 2013 - Abstracts der 45. Arbeitstagung

2013-08 *Daniel Ott, Dr. Alexander Raschke*
Evaluating Benefits of Requirement Categorization in Natural Language Specifications for Review Improvements

2013-09 *Philip Geiger, Rüdiger Pryss, Marc Schickler, Manfred Reichert*
Engineering an Advanced Location-Based Augmented Reality Engine for Smart Mobile Devices

2014-01 *Andreas Lanz, Manfred Reichert*
Analyzing the Impact of Process Change Operations on Time-Aware Processes

2014-02 *Andreas Steigmiller, Birte Glimm, and Thorsten Liebig*
Coupling Tableau Algorithms for the DL SROIQ with Completion-based Saturation Procedures

2014-03 *Thomas Geier, Felix Richter, Susanne Biundo*
Conditioned Belief Propagation Revisited: Extended Version

Ulmer Informatik-Berichte
ISSN 0939-5091

Herausgeber:
Universität Ulm
Fakultät für Ingenieurwissenschaften und Informatik
89069 Ulm