Completion Graph Caching Extensions and Applications for Expressive Description Logics

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. Tableau-based reasoning systems are primarily used for reasoning with knowledge bases that are based on very expressive Description Logics such as $\mathcal{SROIQ}$ since they enable a straightforward handling of non-deterministic language features such as disjunctions and cardinality restrictions. However, these tableau-based reasoners are often not very efficient for large ABoxes, i.e., knowledge bases with many facts, because they have to repeatedly re-consider the consequences of the ABoxes for many reasoning tasks. In this paper, we present a refined completion graph caching technique, where the (non-deterministically) derived consequences in the consistency test of the ABox are cached such that the subsequent reasoning effort w.r.t. the ABox can be reduced. In addition, we present several extensions and applications of the caching technique, which, for example, enable incremental reasoning for changing ABoxes. The presented techniques are implemented in our reasoning system Konclude and our evaluation shows that they significantly improve the reasoning performance.

1 Introduction

The current version of the Web Ontology Language (OWL 2) [29], standardised by the World Wide Web Consortium (W3C), is based on the Description Logic (DL) $\mathcal{SROIQ}$, which supports very expressive language features such as qualified cardinality restrictions, inverse roles, role chains, and nominals [13]. The latter one allow for referring individuals directly within concept expressions to, for example, define a concept as a singleton with only the specified individual as member.

Whereas DL-based knowledge bases have often been differentiated in a terminological part (TBox), where the vocabulary and the relationships between terms are specified, and an assertional part (ABox), containing the facts about concrete individuals, such a strict separation is not longer possible with nominals. In particular, due to the nominals, both parts depend on each other, which makes reasoning in practice often more difficult. For example, classification is the well-known reasoning task where we are interested in the subsumption hierarchy of the atomic concepts of a knowledge base, which could originally be calculated for nominal-free knowledge bases by only considering the axioms in the TBox (besides of an inconsistent ABox, for which all (atomic) concepts are equivalent to bottom). With nominals, also the ABox has to be considered for classification, whereby the reasoning time often increases dramatically.

In addition, knowledge bases that use language features of more expressive DLs often contain non-determinism, e.g., due to disjunctions or cardinality restrictions. As a consequence, often more sophisticated reasoning procedures are required, which are able to do case-by-case analyses. The current state-of-the-art reasoners for very expressive DLs such as $\mathcal{SROIQ}$ are typically based on variants of tableau algorithms, refutation-based model construction calculi, where higher level reasoning tasks are reduced to possibly many consistency tests. Obviously, if it is necessary to consider the ABox for each of these consistency tests, then the reasoning easily becomes infeasible in practice for knowledge bases that have large ABoxes and use more expressive language features.

For less expressive DLs, already several optimisations have been proposed that enable a reduction of the ABox reasoning effort especially for the instance checking problem [30,31], i.e., these optimisations allow for considering only parts of the ABox for checking whether an individual is an instance of a specific concept. However, it is not clear how these optimisations can be extended to $\mathcal{SROIQ}$ and they often bear several disadvantages. For example, optimisation approaches based on partitioning and modularisation techniques require a syntactic pre-analysis of the concepts, roles, and individuals in a knowledge base, which can be, particularly for more expressive DLs, quite costly, and, due to the static analysis, only queries with specific concepts are supported. If axioms of the knowledge base are extended and modified such that the ABox reasoning is improved, then there exists the risk that this negatively influences other reasoning tasks for which ABox reasoning is potentially less relevant.

Another possibility to avoid the redundant re-processing of the ABox is the caching of the data from the initial consistency check. Tableau algorithms usually construct a so-called completion graph for a consistency check, which can be re-used in subsequent tests to reduce the number of consequences that have to be re-derived for the individuals of the ABox. Existing approaches based on this idea (e.g., [22]) are, however, not very suitable in handling non-determinism. In particular, the caching and re-use of non-deterministically derived facts can easily cause unsound consequences and, therefore, the non-deterministically derived facts of the cached completion graph are usually simply discarded and, if necessary, re-derived as soon as they could potentially be problematic. We address this by presenting a refinement of caching idea, where we check with a set of conditions which non-deterministic parts of a cached completion graph can safely be re-used. The caching conditions can be locally checked and, thus, they allow for efficiently identifying individuals step by step for which non-deterministic consequences have to be re-considered in subsequent tests. The presented technique can be directly integrated into existing tableau-based reasoning systems without significant adaptations and reduces ABox reasoning automatically for all reasoning tasks for which consequences from the ABox are potentially relevant. Moreover, it can be directly used for the DL $\mathcal{SROIQ}$, does not produce a significant overhead, and can easily be extended, for example, to improve incremental ABox reasoning.

The reminder of the paper is organised as follows: We first introduce some preliminaries in Section 2, then we present the basic completion graph caching technique in Section 3. After that, we present several extensions and applications of the caching approach in Section 4, which allow for supporting nominals in ordinary satisfiability caching techniques (Section 4.1), improving incremental reasoning for changing ABoxes (Section 4.2), and handling very large ABoxes by storing data in a representative way (Section 4.3). In Section 5, we discuss similarities and differences to related work in more detail. Finally, we present a detailed evaluation in Section 6 before we conclude in Section 7.

2 Preliminaries

Since our technique is designed to support all language features of the DL $\mathcal{SROIQ}$, we first give a brief introduction into $\mathcal{SROIQ}$ (see [1] for a detailed introduction into DLs), and then we present a tableau algorithm for $\mathcal{SROIQ}$ to which the technique can be directly integrated (see [13] 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 $a \in N_I, n \in \mathbb{N}_0, A \in N_C$, 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* role inclusion *(RI) axiom is an expression of the form $u \sqsubseteq r$, where r is a role and u is a composition of roles, i.e., $u = s_1 \circ \ldots \circ s_n$ with the roles $s_1, \ldots, s_n$ and $n \geq 1$. For r, s roles, a* role assertion *(RA) axiom is of the form $\mathsf{Disj}(r, s)$ or $\mathsf{Refl}(r)$. An RBox is a finite set of RIs and RAs. 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 types of RAs for 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, as described below in more detail.

Definition 3 (Regularity of Role Inclusion Axioms). *A set of RIs is* regular *if the used role names can be sorted in a strict partial order and all RIs are regular. Let $\prec$ be a strict partial order on role names, then for the role names $r, s_1, \ldots, s_n$, the RI 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 RI axioms, we call a role* simple *if it is not complex. A role r is called* complex *w.r.t. a set of RIs if*

1. *its inverse role is complex, or*
2. *it occurs on the right-hand side of a RI 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:

$$\begin{array}{rlrlrl}
\top^\mathcal{I} = & \Delta^\mathcal{I} & \bot^\mathcal{I} = & \emptyset & (\{a\})^\mathcal{I} = & \{a^\mathcal{I}\} \\
(\neg C)^\mathcal{I} = & \Delta^\mathcal{I} \setminus C^\mathcal{I} & (C \sqcap D)^\mathcal{I} = & C^\mathcal{I} \cap D^\mathcal{I} & (C \sqcup D)^\mathcal{I} = & C^\mathcal{I} \cup D^\mathcal{I} \\
(\exists r.\mathsf{Self})^\mathcal{I} = & \multicolumn{5}{l}{\{\delta \in \Delta^\mathcal{I} \mid \langle \delta, \delta \rangle \in r^\mathcal{I}\}} \\
(\forall r.C)^\mathcal{I} = & \multicolumn{5}{l}{\{\delta \in \Delta^\mathcal{I} \mid \text{if } \langle \delta, \delta' \rangle \in r^\mathcal{I}, \text{ then } \delta' \in C^\mathcal{I}\}} \\
(\exists r.C)^\mathcal{I} = & \multicolumn{5}{l}{\{\delta \in \Delta^\mathcal{I} \mid \text{there is a } \langle \delta, \delta' \rangle \in r^\mathcal{I} \text{ with } \delta' \in C^\mathcal{I}\}} \\
(\leqslant n\, r.C)^\mathcal{I} = & \multicolumn{5}{l}{\{\delta \in \Delta^\mathcal{I} \mid \sharp\{\delta' \in \Delta^\mathcal{I} \mid \langle \delta, \delta' \rangle \in r^\mathcal{I} \text{and } \delta' \in C^\mathcal{I}\} \leq n\}} \\
(\geqslant n\, r.C)^\mathcal{I} = & \multicolumn{5}{l}{\{\delta \in \Delta^\mathcal{I} \mid \sharp\{\delta' \in \Delta^\mathcal{I} \mid \langle \delta, \delta' \rangle \in r^\mathcal{I} \text{and } \delta' \in C^\mathcal{I}\} \geq n\},}
\end{array}$$

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

$\mathcal{I}$ satisfies *a TBox $\mathcal{T}$ (RBox $\mathcal{R}$, ABox $\mathcal{A}$) if it satisfies each GCI in $\mathcal{T}$ (each RI/RA axiom in $\mathcal{R}$, each assertion in $\mathcal{A}$). We say that $\mathcal{I}$* satisfies *$\mathcal{K}$ 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 [16]:

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

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 have 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 axioms 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 RIs 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 RI $u \sqsubseteq r \in R$ 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 RI $u \sqsubseteq r \in R$ 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 $\forall r.C$ be an universal restriction that occurs in a knowledge base $\mathcal{K}$. We say that $\forall r.C$ is normalised *w.r.t. $\mathcal{K}$ if*

- *C is an atomic concept, $\forall r.C$ 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 algorithm 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 blow up the knowledge base exponentially. Although such a blow up cannot be avoided in the worst-case [18], 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 [13]. 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 $G = (V, E, \mathcal{L}, \dot{\neq}, \mathcal{M})$. 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 and the mapping $\mathcal{M}$ is used to store the merging history for nodes.

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, $\mathsf{mneighbs}^G(v, r, C)$ 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 $G = (V, E, \mathcal{L}, \dot{\neq}, \mathcal{M})$ 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* $\{w_1, \ldots, w_{n+1}\} \subseteq \mathsf{mneighbs}^G(v, r, C)$ *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* [14] 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* $\mathcal{L}(v') = \mathcal{L}(w')$*,*
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 extend $\mathcal{M}$ by $\{w \mapsto v\}$ and we "prune" the completion graph by removing w as well as, 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* $G = (V, E, \mathcal{L}, \dot{\neq}, \mathcal{M})$*, written* $\mathsf{prune}(w)$*, yields a graph that is obtained from* G *as follows:*

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

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

1. *add* $\{w \mapsto v\}$ *to* $\mathcal{M}$
2. *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*;*

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. $\sharp\mathsf{mneighbs}^G(v, r, C) > n$ 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 $1 \leq i < j \leq m$.

3. *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;*
4. *$\mathcal{L}(v) = \mathcal{L}(v) \cup \mathcal{L}(w)$;*
5. *add $v \dot{\neq} v'$ for all v' such that $w \dot{\neq} v'$;*
6. $\mathsf{prune}(w)$.

For a node v and a node mapping $\mathcal{M}$, we use $\mathsf{mergedTo}^{\mathcal{M}}(v)$ to denote the function that returns the node into which v has been merged as follows:

$$\mathsf{mergedTo}^{\mathcal{M}}(v) = \begin{cases} \mathsf{mergedTo}^{\mathcal{M}}(w) & \text{if } v \mapsto w \in \mathcal{M}, \\ v & \text{else}. \end{cases}$$

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 [17] 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 [24,25], are further improving the handling of knowledge bases for more expressive DLs since the non-determinism that is caused by disjunctions on the right-hand side of axioms is further reduced. Roughly speaking, the non-absorbable disjuncts are partially used as conditions on the left-hand side of additional inclusion axioms such that the processing of the disjunctions can further be delayed.

Many state-of-the-art reasoning systems are at least using some kind of binary absorption, which makes the processing of simple ontologies (e.g., EL ontologies) also with the tableau algorithm deterministic. In the following, we assume that knowledge bases are at least preprocessed with such 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 idea with 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 with lazy unfolding rules.

3 Completion Graph Caching and Reusing

Higher level reasoning tasks are usually reduced to consistency checking for tableau-based reasoning systems. For instance, checking a subsumption between two atomic concepts A and B, as required during classification, i.e., the reasoning task to determine the subsumption hierarchy of all atomic concepts in a knowledge base, can simply be reduced to checking whether $A \sqcap \mathsf{norm}(\neg B)$ is satisfiable. For this, it is checked whether the knowledge base, extended with a new individual for which $A \sqcap \mathsf{norm}(\neg B)$ is asserted, is consistent. In principle, all individuals in the knowledge base have to be considered in these consistency tests and if a knowledge base contains many individuals, then the tableau algorithm could be forced to repeatedly apply the expansion rules for them, thus doing very similar work very often. Clearly, if there is no interaction between the original individuals in the knowledge base and the new ones that are temporarily created for checking the satisfiability of certain constructs, then it is not necessary to create and process the nodes for the original individuals in a corresponding completion graph since the tableau algorithm could expand them in the same way as for the initial consistency test of the knowledge base. However, for knowledge bases that use more expressive language features such as nominals and/or restrictions w.r.t. the universal role, such an interaction cannot easily be excluded. Moreover, some reasoning tasks such as instance checking require the direct modification of the original individuals. For instance, to check whether the individual a is an instance of the concept A, we have to test the consistency of a knowledge base that is extended by the assertion $\mathsf{norm}(\neg C)(a)$. Hence, a technique that reduces redundant work for not affected individuals in subsequent consistency tests is useful or even necessary to handle expressive and large ontologies with many individuals.

One possible technique to avoid the redundant re-processing of individuals is the caching and re-use of data from the completion graph of the initial consistency check. Existing approaches based on this idea (e.g., [22]) are, however, not very suitable in handling non-determinism. We address this by presenting a refinement of this caching technique in the following, which also allows for re-using non-deterministic parts of the cached completion graph such that only the nodes for those individuals have to re-processed that are indeed affected by new consequences in subsequent consistency tests.

To present our completion graph caching technique in the following, we distinguish different versions of completion graphs with superscripts, e.g., by the application of an expansion rule to an initial completion graph G^0, we obtain a completion graph G^1 that is a modification of G^0 according to the rule. If the rule is non-deterministic, then we obtain for each alternative a separate completion graph.

In the following, we denote with $G^d = (V^d, E^d, \mathcal{L}^d, \dot{\neq}^d, \mathcal{M}^d)$ the last completion graph of the initial consistency test that is obtained with only deterministic rule applications, and with $G^n = (V^n, E^n, \mathcal{L}^n, \dot{\neq}^n, \mathcal{M}^n)$ the fully expanded (and clash-free) completion graph that possibly also contains non-deterministic choices. Obviously, instead of starting with G^0, we can use G^d to initialise a completion graph G for subsequent consistency tests which are, for example, required to prove or refute assumptions of higher level reasoning tasks. To be more precise, we extend G^d to G by the new individuals or by additional assertions to original individuals as required for a subsequent test and then we can apply the tableau expansion rules to G. Note, in order to be able to distinguish the nodes/nominals in the different completion graphs, we assume that all nodes/nominals that are newly created for G do not occur in existing completion graphs, such as $G^d, \ldots, G^n$.

This re-use of G^d is an obvious and straightforward optimisation and, therefore, it is already used by many state-of-the-art reasoning systems to successfully reduce the work that has to be repeated in subsequent consistency tests [22]. Especially if the knowledge base is deterministic, then the tableau expansion rules only have to be applied for the newly added assertions in G. In principle, also G^n can be re-used instead of G^d [22], but this causes problems if non-deterministically derived facts of G^n are involved in new clashes. In particular, it is required to do backtracking in such cases, i.e., we have to jump back to the last version of the initial completion graph that does not contain the consequences of the last non-deterministic decision that is involved in the clash. Then, we have to continue the processing by choosing another alternative. Obviously, if we have to jump back to a very early version of the completion graph, then potentially many non-deterministic decisions must be re-processed. Moreover, after jumping back, we also have to add and re-process the newly added individuals and/or assertions.

To improve the handling of non-deterministic knowledge bases, our approach uses criteria to check whether nodes in G (or the nodes in a completion graph G' obtained from G by further rule applications) are "cached", i.e., there exist corresponding nodes in the cached completion graph of the initial consistency test and, therefore, it is not required to process them again. To be more precise, these caching criteria check whether the expansion of nodes is possible as in the cached completion graph G^n without influencing modified nodes in G', thus only the processing of new and modified individuals is required.

Definition 14 (Caching Criteria). *Let $G^d = (V^d, E^d, \mathcal{L}^d, \dot{\neq}^d, \mathcal{M}^d)$ be a completion graph with only deterministically derived consequences and $G^n = (V^n, E^n, \mathcal{L}^n, \dot{\neq}^n, \mathcal{M}^n)$ a completion graph that contains the consequences of a fully expanded and clash-free expansion of G^d. Moreover, let G be an extension of G^d that is obtained by adding nodes and/or concepts to node labels and let $G' = (V', E', \mathcal{L}', \dot{\neq}', \mathcal{M}')$ be a completion graph that is obtained from G by rule applications.*

A node $v' \in V'$ is cached in G' w.r.t. G^d and G^n *if caching of the node is not invalid, where the* caching is invalid *(we then also refer to the node as non-cached) if*

C1 $v' \notin V^d$ *or* $\mathsf{mergedTo}^{\mathcal{M}^n}(v') \notin V^n$*;*

C2 $\mathcal{L}'(v') \not\subseteq \mathcal{L}^n(\mathsf{mergedTo}^{\mathcal{M}^n}(v'))$*;*

C3 $\forall r.C \in \mathcal{L}^n(\mathsf{mergedTo}^{\mathcal{M}^n}(v'))$ *and there is an r-neighbour node w' of v' such that w' is not cached and $C \notin \mathcal{L}(w')$;*

C4 $\leqslant m\, r.C \in \mathcal{L}^n(\mathsf{mergedTo}^{\mathcal{M}^n}(v'))$ *and there is a non-cached r-neighbour node w' of v' with $\{C, \mathsf{nnf}(\neg C)\} \not\subseteq \mathcal{L}'(w')$;*

C5 $\leqslant m\, r.C \in \mathcal{L}^n(\mathsf{mergedTo}^{\mathcal{M}^n}(v'))$ *and the number of the non-cached r-neighbours of v' with C in their labels together with the r-neighbours in G^n of* $\mathsf{mergedTo}^{\mathcal{M}^n}(v')$ *with C in their labels is greater than m;*

C6 $\exists r.C \in \mathcal{L}^n(\mathsf{mergedTo}^{\mathcal{M}^n}(v'))$ *and there is no r-neighbour w' of v' with $C \in \mathcal{L}(w')$ and every r-neighbour w' of v' with $C \in \mathcal{L}^n(\mathsf{mergedTo}^{\mathcal{M}^n}(w'))$ is not cached;*

C7 $\geqslant m\, r.C \in \mathcal{L}^n(\mathsf{mergedTo}^{\mathcal{M}^n}(v'))$ *and the number of r-neighbour nodes $w^n_1, \ldots, w^n_k$ of* $\mathsf{mergedTo}^{\mathcal{M}^n}(v')$*, for which it holds, for $1 \leq i < j \leq k$, that $C \in \mathcal{L}^n(w^n_i)$, $\mathcal{L}^n(w^n_i) \dot{\neq}^n \mathcal{L}^n(w^n_j)$, and there is no node $w'_i \in V'$ with* $\mathsf{mergedTo}^{\mathcal{M}^n}(w'_i) = w^n_i$ *or w'_i with* $\mathsf{mergedTo}^{\mathcal{M}^n}(w'_i) = w^n_i$ *and $C \notin \mathcal{L}(w'_i)$ is not cached, is less than m;*

C8 $\mathsf{mergedTo}^{\mathcal{M}^n}(v')$ *is a nominal node with $\leqslant m\, r.C$ in its label and there exists a blockable and non-cached predecessor node w' of v' that is* $\mathsf{inv}(r)$*-neighbour of v' and does not have the concept* $\mathsf{nnf}(\neg C)$ *in its label;*

C9 $\mathsf{mergedTo}^{\mathcal{M}^n}(w^d)$ *is an r-neighbour of* $\mathsf{mergedTo}^{\mathcal{M}^n}(v')$ *such that* $\mathsf{mergedTo}^{\mathcal{M}'}(w^d)$ *is not cached and* $\mathsf{mergedTo}^{\mathcal{M}'}(w^d)$ *is not an r-neighbour node of v';*

C10 $\mathsf{mergedTo}^{\mathcal{M}^n}(v')$ *has a neighbour u^n_1 such that there is a path of neighboured nodes $u^n_1, \ldots u^n_k$ with $k \geq 1$, for which there is no node $u'_i \in V'$ with* $\mathsf{mergedTo}^{\mathcal{M}^n}(u'_i) = u^n_i$ *for $1 \leq i \leq k$, where u^n_k has a neighbour node* $\mathsf{mergedTo}^{\mathcal{M}^n}(w^d)$ *for which* $\mathsf{mergedTo}^{\mathcal{M}'}(w^d) \in V'$ *and* $\mathsf{mergedTo}^{\mathcal{M}'}(w^d)$ *is not cached;*

C11 $\mathsf{mergedTo}^{\mathcal{M}^n}(v')$ *is blocked by* $\mathsf{mergedTo}^{\mathcal{M}^n}(w^d)$ *and* $\mathsf{mergedTo}^{\mathcal{M}'}(w^d) \in V'$ *is non-cached;*

C12 *there is a non-cached node* $\mathsf{mergedTo}^{\mathcal{M}'}(w^d) \in V'$ *such that* $\mathsf{mergedTo}^{\mathcal{M}^n}(w^d) = \mathsf{mergedTo}^{\mathcal{M}^n}(v')$*; or*

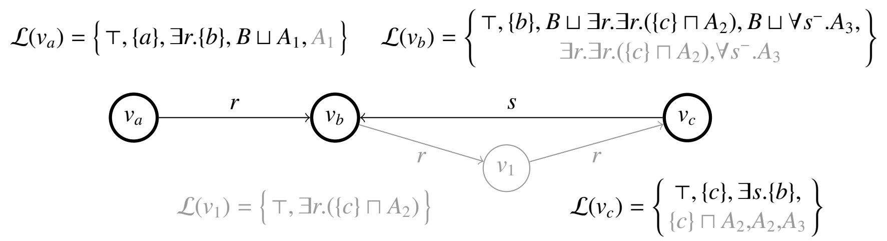
C13 *there is a node* $w' \in V'$ *such that* $v' \dot{\neq}' w'$ *and* $\mathsf{mergedTo}^{\mathcal{M}^n}(v') = \mathsf{mergedTo}^{\mathcal{M}^n}(w')$.

Please note that we allow G^n to be an extension of a fully expanded and clash-free completion graph, i.e., G^n must contain the consequences of a fully expanded, clash-free completion graph, but, beyond that, we allow G^n to also contain additional consequences that potentially would even result in clashes. This is useful to simplify extensions of the caching technique (cf. Section 4) and is not problematic for the construction of subsequent completion graphs since the caching criteria ensure that the cached parts, which contain at least the consequences of fully expanded and clash-free completion graph, can be re-used without causing new clashes. Also note that we only allow G to be an extension of G^d where new nodes are added or node labels are extended by new concepts since the completion graph caching requires changes in node labels to identify parts that have to be processed. In particular, if we want to test whether two individuals are the same, then we cannot simply create a satisfiability test where $\dot{\neq}^d$ is extended, but we have to add the negated nominal of one individual to the node label of the other individual.

Conditions C1 and C2 ensure that a node also exists in the cached completion graph G^n and that its label is a subset of the corresponding label in G^n such that the same expansion is possible. C3 checks whether the expansion of a node would add a concept of the form $\forall r.C$ such that it could propagate C to a non-cached neighbour node. Analogously, C4 and C5 check for potentially violated at-most cardinality restrictions by counting the new or modified neighbours in G' and the neighbours in G^n. C6 and C7 verify that existential and at-least cardinality restrictions are still satisfied if the cached nodes are expanded identically. C8 checks whether the NN-rule of the tableau algorithm would be applicable after the expansion, i.e., we check whether all potentially relevant neighbours in G' are nominal nodes. C9 checks whether the expansion would add additional roles to edge labels between cached and non-cached nodes, which could be problematic for disjoint roles. For C10: If a node w', for which caching is invalid, is connected to nodes in G^n that are only available in G^n (e.g., non-deterministically created ones), then we have to identify caching as invalid for those ancestors of these nodes that are also in G' such that these connections to w' can be re-built. Otherwise, we would potentially miss new consequences that could be propagated from w'. C11 is used to reactivate the processing of nodes for which the caching of the blocker node is invalid. C12 and C13 ensure that merging is possible as in G^n: C12 checks whether the node into which the node is merged is also cached and C13 ensures that there is no additional entry for $\dot{\neq}'$ that would cause a clash if the nodes were merged as in G^n.

Note that some of the conditions are more restrictive than necessary in order to keep the caching tests simple and efficient. For instance, instead of C10, we could check more precisely whether the new concepts in w' could influence the connected nodes in G^n. We assume, however, that refinements of C10 or C9 do usually not lead to significant improvements since it can often achieved in practice (through adequate absorption techniques) that the processing of ontologies is primarily deterministic. Analogously, we could also simply use a condition for at most cardinality restrictions of the form $\leqslant m\, r.C$, where we only count how many of the new or modified non-cached neighbour nodes do not have $\mathsf{nnf}(\neg C)$ in their label (instead of using C4 and C5), since either $\neg C$ or C is satisfiable, and then also the cardinality restriction is trivially satisfied if the

Fig. 1 Constructed and cached completion graph for Example 2 with deterministically (coloured black) and non-deterministically (coloured grey) derived facts


$\mathcal{L}(v_a) = \{\top, \{a\}, \exists r.\{b\}, B \sqcup A_1, A_1\}$
$\mathcal{L}(v_b) = \{\top, \{b\}, B \sqcup \exists r.\exists r.(\{c\} \sqcap A_2), B \sqcup \forall s^-.A_3, \exists r.\exists r.(\{c\} \sqcap A_2), \forall s^-.A_3\}$
v_a
r
v_b
s
v_c
r
v_1
r
$\mathcal{L}(v_1) = \{\top, \exists r.(\{c\} \sqcap A_2)\}$
$\mathcal{L}(v_c) = \{\top, \{c\}, \exists s.\{b\}, \{c\} \sqcap A_2, A_2, A_3\}$

number of such neighbour nodes together with the relevant neighbours from the cached completion graph is less than m. However, this would complicate the correctness proof (cf. Section 3.1 significantly since it would not be directly possible to construct a fully expanded and clash-free completion graph from the G', where one could show that the tableau expansion rules are not further applicable (in particular, the applicability of the ch-rule could be problematic).

Please also note that the cached nodes cannot be used for blocking. In particular, the cached nodes are incompletely processed and if we use them to block the expansion of other nodes, then we potentially miss some consequences. Also the nodes in G^n cannot be used for blocking since new consequences in G', e.g., concepts propagated over nominals, are not considered in G^n.

In order to maximise the caching, it is important to first process the completion graph deterministically as much as possible. In particular, only those nodes can be cached which are also available in the "deterministic" completion graph G^d. On one hand, this can, for example, be achieved by processing the completion graph with deterministic rules only until the generation of new nodes is subset blocked. Subset blocking is not sufficient for more expressive DLs (inverse roles and cardinality restrictions require pairwise blocking), but it prevents the expansion of too many successor nodes in case non-deterministic rule applications merge and prune some parts of the completion graph. On the other hand, absorption techniques (e.g., [17,25,27]) are essential since they reduce the overall non-determinism in the knowledge base.

Example 2. Let us assume that the tableau algorithm builds a completion graph as depicted in Figure 1 for testing the consistency of a knowledge base $\mathcal{K}$ containing the axioms

$$\{a\} \sqsubseteq B \sqcup A_1 \qquad \{b\} \sqsubseteq B \sqcup \exists r.\exists r.(\{c\} \sqcap A_2) \qquad \{c\} \sqsubseteq \exists s.\{b\}$$
$$\{a\} \sqsubseteq \exists r.\{b\} \qquad \{b\} \sqsubseteq B \sqcup \forall s^-.A_3$$

We refer to the deterministic version of the completion graph as G^d (represented by those elements in Figure 1 that are coloured in black) and with G^n to the non-deterministic version (consisting of the elements coloured in black as well as grey in Figure 1).

If we now want to determine which individuals are instances of the concept $\exists r.\top$, then we have to check, for each individual i in $\mathcal{K}$, the consistency of $\mathcal{K}$ extended by

$\mathsf{norm}(\neg\exists r.\top)(i)$, which is equivalent to adding the axiom $\{i\} \sqsubseteq \forall r.\bot$. The individual a is obviously an instance of this concept, which can also be observed by the fact that expanding the completion graph adds $\forall r.\bot$ to $\mathcal{L}^d(v_a)$, which immediately results in a clash. In contrast, if we extend $\mathcal{L}^d(v_b)$ by $\forall r.\bot$, we have to process the disjunctions $B \sqcup \exists r.\exists r.(\{c\} \sqcap A_2)$ and $B \sqcup \forall s^-.A_3$. The disjunct $\exists r.\exists r.(\{c\} \sqcap A_2)$ would result in a clash and, therefore, we choose B, which also satisfies the second disjunction. Note that v_a and v_c do not have to be processed since v_a is cached, i.e., its expansion is possible in the same way as in G^n, and v_c does not have any concepts for which processing is required. Last but not least, we have to test whether $\mathcal{L}^d(v_c)$ extended by $\forall r.\bot$ is satisfiable. Now, the caching of v_c is obviously invalid (due to C2) and, therefore, also the caching of v_b is invalid: C3 can be applied for $\forall s^-.A_3$ and C10 for the successor node v_1 of v_b in G^n which also has the non-cached successor node v_c. Since v_a is still cached, we only have to process v_b again, which does, however, not result in a clash. Hence, only a is an instance of $\exists r.\top$.

It is worth pointing out that the majority of all conditions can be checked locally. If the caching of a node is identified as invalid, then we simply follow the edges w.r.t. G' and G^n to those (potential indirect) neighbour nodes that are also available in G^d. Minor exceptions are Conditions C11, C12, and C13, for which we can, however, simply trace back established blocking relations or $\mathcal{M}^n$ to find nodes for which the caching criteria have to be checked. Instead of directly checking the caching criteria, the relevant nodes can also be stored in a set to check the conditions when it is (more) convenient.

It is clear that the satisfaction of some conditions can change after the applications of rules. For example, if we have cached a completion graph where v has the non-deterministically derived concept $\forall r.C$ in its label and we now process a new completion graph with a node w that is an r-neighbour of v and has $C \sqcup D$ in its label, then we also have to test whether the caching of v is invalid for the new completion graph. If the disjunction is processed first by adding C to the label of w, then none of the conditions can be applied to v. In contrast, if we first were to check the caching criteria for v, then Condition C3 would be satisfied due to the concept $\forall r.C$ that could propagate C to w. As a consequence, the caching of v would be invalid and we would have to schedule v for processing. Hence, if the caching criteria are checked later, then potentially fewer nodes have to be processed. However, delaying the checks for nodes that can potentially be cached can also have significant disadvantages. In particular, if non-deterministic choices for the potentially cached individuals are often involved in clashes, then the caching criteria should be checked as soon as possible. Even if a wrong non-deterministic alternative is chosen first, then the clash can often be discovered before processing many other nodes and we can immediately jump back and change a relevant choice (e.g., with dependency directed backtracking [1,28]). In contrast, if we were to process as many nodes as possible before checking the caching criteria, then we would potentially do a lot of work before the processing of the potentially cached nodes would be activated, and by jumping back to an early non-deterministic decision, it could be necessary to repeat this work (or very similar work) quite often. Thus, if the caching criteria are checked early, then the tableau algorithm can schedule the processing of nodes as usual, which also allows for prioritising the processing of those nodes and/or concepts for which there is a high likelihood that they are involved in clashes.

Clearly, an optimal strategy for the determination of the point of time where the caching criteria have to be checked depends on the ontology. However, it can often be observed that many ontologies are primarily deterministic and that non-deterministic consequences have only a locally limited influence and, for such ontologies, both strategies should work reasonably well. Alternatively, one could use a strategy that learns for the entire knowledge base or for single nodes whether the criteria should be checked earlier or later. Unfortunately, this cannot easily be realised in many reasoning systems since dependencies between derived facts in the completion graph are often not tracked very precisely and, therefore, it cannot be detected which nodes are often involved in the creation of clashes such that the strategy could be updated to prioritise the checking of the caching criteria for these nodes.

It is also possible to non-deterministically re-use the derived consequences from G^n, i.e., if the caching is invalid for a node v and $\mathsf{mergedTo}^{\mathcal{M}^n}(v)$ is in G^n, then we can non-deterministically add the missing concepts from $\mathcal{L}^n(\mathsf{mergedTo}^{\mathcal{M}^n}(v))$ to $\mathcal{L}(v)$. This can be used to build a completion graph that is very similar to the cached one and, as a consequence, caching can often quickly be established for many nodes. Of course, if some of the non-deterministically added concepts are involved in clashes, then we potentially have to backtrack and process the alternative where this node is ordinarily processed. One can further learn statistics for the re-use of nodes in order to avoid too much backtracking, which can be caused by non-deterministically re-used concepts that are often involved in clashes.

The presented completion graph caching obviously requires that G^d as well as G^n are both kept in memory, which can be a significant overhead for knowledge bases that contain many individuals. However, state-of-the-art reasoning systems already store G^d and they typically use an approach to share data between completion graphs if only some parts are modified. Hence, G^n does usually not significantly increase the memory consumption compared to G^d as long as there are not too many non-deterministic consequences. Moreover, with the completion graph caching it is often sufficient to only re-process smaller parts of the initially cached completion graph and, therefore, subsequent consistency tests do usually not increase the overall memory consumption of the reasoning system significantly.

A nice side effect of storing G^n is that we can use the non-deterministic decisions from G^n as an orientation in subsequent consistency tests. In fact, if we prioritise the processing of the same non-deterministic alternatives as for G^n, then we can potentially find a solution that is very similar to G^n without exploring much of the search space.

3.1 Correctness

It is obvious that the presented completion graph caching technique does not influence termination of the tableau algorithm. Moreover, it is not possible that a wrong clash can be discovered since only the processing of existing nodes is blocked. Therefore, we can focus on proving completeness, i.e., we have to show that a fully processed completion graph with cached nodes (G') can be expanded to a model. For this, it is also sufficient to show that the cached nodes can be expanded as in the cached completion graph such that we obtain a fully expanded and clash-free completion graph. In order to show this in the following, we first define how a fully expanded and clash-free completion graph, say G, can be build from G' by expanding the cached nodes of G' as for G^n.

Definition 15 (Completion Graph Expansion). *Let $G^d = (V^d, E^d, \mathcal{L}^d, \dot{\neq}^d, \mathcal{M}^d)$ be a completion graph with only deterministically derived consequences and $G^n = (V^n, E^n, \mathcal{L}^n, \dot{\neq}^n, \mathcal{M}^n)$ a completion graph that contains the consequences of a fully expanded and clash-free expansion of G^d. Moreover, let $G' = (V', E', \mathcal{L}', \dot{\neq}', \mathcal{M}')$ be a clash-free extension of G^d that is obtained by adding new nodes to the completion graph and/or new concepts to node labels and by applying tableau expansion rules such that all non-cached nodes are fully expanded, i.e., tableau expansion rules cannot further be applied to non-cached nodes. The* expanded completion graph $G = (V, E, \mathcal{L}, \dot{\neq}, \mathcal{M})$ *is obtained from G' by setting:*

- $V = V_p \cup V_e$, *where V_p denotes the set of* processed nodes, *i.e.,*
  $V_p = \{v \in V' \mid v \text{ is not cached }\}$,
  and V_e denotes the set of expanded nodes, *i.e.,*
  $V_e = \{v^n \in V^n \mid$ *there is no node* $\mathsf{mergedTo}^{\mathcal{M}'}(v^d) \in V'$ *such that*
  $v^n = \mathsf{mergedTo}^{\mathcal{M}^n}(v^d)$ *or* $\mathsf{mergedTo}^{\mathcal{M}'}(v^d)$ *is cached* $\}$.
- $E = \{\langle v, w\rangle \in E' \mid \{v, w\} \subseteq V_p\} \cup$
  $\{\langle \mathsf{mergedTo}^{\mathcal{M}^n}(v), w\rangle \mid \langle v, w\rangle \in E' \wedge \mathsf{mergedTo}^{\mathcal{M}^n}(v) \in V_e \wedge w \in V_p\} \cup$
  $\{\langle v, \mathsf{mergedTo}^{\mathcal{M}^n}(w)\rangle \mid \langle v, w\rangle \in E' \wedge v \in V_p \wedge \mathsf{mergedTo}^{\mathcal{M}^n}(w) \in V_e\} \cup$
  $\{\langle v, w\rangle \in E^n \mid \{v, w\} \cap V_p = \emptyset\}$.
- $\mathcal{L} = \{\ell \in \mathcal{L}'(v) \mid v \in V_p\} \cup$
  $\{\ell \in \mathcal{L}^n(v) \mid v \in V_e\} \cup$
  $\{\ell \in \mathcal{L}'(\langle v, w\rangle) \mid \{v, w\} \subseteq V_p\} \cup$
  $\{\langle \mathsf{mergedTo}^{\mathcal{M}^n}(v), w\rangle \mapsto \mathcal{R} \mid \langle v, w\rangle \mapsto \mathcal{R} \in \mathcal{L}' \wedge$
  $\mathsf{mergedTo}^{\mathcal{M}^n}(v) \in V_e \wedge w \in V_p\} \cup$
  $\{\langle v, \mathsf{mergedTo}^{\mathcal{M}^n}(w)\rangle \mapsto \mathcal{R} \mid \langle v, w\rangle \mapsto \mathcal{R} \in \mathcal{L}' \wedge$
  $v \in V_p \wedge \mathsf{mergedTo}^{\mathcal{M}^n}(w) \in V_e\} \cup$
  $\{\ell \in \mathcal{L}^n(\langle v, w\rangle) \mid \{v, w\} \cap V_p = \emptyset\}$.
- $\dot{\neq} = \{\langle v, w\rangle \in \dot{\neq}' \mid \{v, w\} \subseteq V_p\} \cup$
  $\{\langle \mathsf{mergedTo}^{\mathcal{M}^n}(v), w\rangle \mid \langle v, w\rangle \in \dot{\neq}' \wedge \mathsf{mergedTo}^{\mathcal{M}^n}(v) \in V_e \wedge w \in V_p\} \cup$
  $\{\langle \mathsf{mergedTo}^{\mathcal{M}^n}(v), w\rangle \mid \langle \mathsf{mergedTo}^{\mathcal{M}^n}(v), \mathsf{mergedTo}^{\mathcal{M}^n}(w)\rangle \in \dot{\neq}^n \wedge$
  $\mathsf{mergedTo}^{\mathcal{M}^n}(v) \in V_e \wedge w \in V_p\} \cup$
  $\{\langle v, \mathsf{mergedTo}^{\mathcal{M}^n}(w)\rangle \mid \langle v, w\rangle \in \dot{\neq}' \wedge v \in V_p \wedge \mathsf{mergedTo}^{\mathcal{M}^n}(w) \in V_e\} \cup$
  $\{\langle v, \mathsf{mergedTo}^{\mathcal{M}^n}(w)\rangle \mid \langle \mathsf{mergedTo}^{\mathcal{M}^n}(v), \mathsf{mergedTo}^{\mathcal{M}^n}(w)\rangle \in \dot{\neq}^n \wedge$
  $v \in V_p \wedge \mathsf{mergedTo}^{\mathcal{M}^n}(w) \in V_e\} \cup$
  $\{\langle \mathsf{mergedTo}^{\mathcal{M}^n}(v), \mathsf{mergedTo}^{\mathcal{M}^n}(w)\rangle \mid \langle v, w\rangle \in \dot{\neq}' \wedge \{v, w\} \cap V_p = \emptyset\}$.
  $\{\langle v, w\rangle \in \dot{\neq}^n \mid \{v, w\} \cap V_p = \emptyset\}$.
- $\mathcal{M} = \emptyset$.

Finally, we prune all nodes in G that are not (indirectly) connected (via the neighbour relation) to a root or a nominal node representing an individual of the knowledge base.

We can assume w.l.o.g. that G^n is clash-free and fully expanded, otherwise we could identify/extract an appropriate subset of G^n for which this assumption is satisfied since G^n contains the consequences of a fully expanded and clash-free completion graph. In addition, we also use the assumption that all edges between cached nodes in G' are labelled with the same roles as in G^n. Again, if this is not the case, then we simply remove those edges and roles that do not occur in G^n. We can now show that G is fully expanded and clash-free:

Lemma 1 (Completeness) *Let G^d, G^n, G', and G be completion graphs as in Definition 15, then the expanded completion graph G is clash-free and fully expanded.*

Proof 1 *Let $G^d = (V^d, E^d, \mathcal{L}^d, \dot{\neq}^d, \mathcal{M}^d)$, $G^n = (V^n, E^n, \mathcal{L}^n, \dot{\neq}^n, \mathcal{M}^n)$, $G' = (V', E', \mathcal{L}', \dot{\neq}', \mathcal{M}')$, and $G = (V, E, \mathcal{L}, \dot{\neq}, \mathcal{M})$ be completion graphs as defined in Definition 15. We first show that G is clash-free. In particular, the clash conditions of Definition 10 are not satisfied, which can be observed from Definitions 14 and 15 as follows:*

- *Clash Conditions 1, 2, and 3 are trivially not satisfied since they only apply on single nodes and they only refer to V, E, and $\mathcal{L}$, for which the data either stems from G' (if a processed node is involved) or from G^n, and both completion graphs are clash-free by assumption.*
- *Clash Condition 4 can only be satisfied for an axiom* $\mathsf{Disj}(r, s)$ *if one processed node, say v, as well as one expanded node, say w^n, is involved. We can also observe that w^n stems from a cached node w', i.e., $w^n = \mathsf{mergedTo}^{\mathcal{M}^n}(w')$, since for other expanded nodes, Condition* C10 *would have invalidated the caching of those ancestors that cause the generation of these expanded nodes and, therefore, they would have been pruned in G. However, Condition* C9 *would identify the caching of w' as invalid if the expansion were to make v an r-neighbour node of w^n and v were not already an r-neighbour node of w' in G'. Hence, the expansion does not add additional roles to edge labels and/or edges between processed and expanded nodes and, therefore, Clash Condition 4 can only be satisfied if v is already an r- and s-neighbour of w' in G', which contradicts our assumption that G' is clash-free.*
- *For Clash Condition 5, we observe that the node v^n with $\leqslant n\, r.C$ in its label must be an expanded node, otherwise the clash would also be in G' (which is contradictory to our assumption) since the definition of E and $\mathcal{L}$ as well as Condition* C9 *guarantee that the expansion does not add/remove roles to edge labels and/or edges between processed and expanded nodes and since it is ensured by Condition* C2 *and the definition of V and $\dot{\neq}$ that the expansion of cached neighbour nodes does neither remove C from their labels nor entries for $\dot{\neq}$. Moreover, v^n must stem from a cached node v', i.e., $v^n = \mathsf{mergedTo}^{\mathcal{M}^n}(v')$, since all other expanded nodes get the same neighbours as in G^n for which the clash condition is not satisfied by assumption. Analogously to Clash Condition 4, we can, however, argue that Condition* C5 *would identify the caching of v' as invalid if the addition of the same r-neighbours to v' as in G^n could violate the cardinality restriction. Hence, Clash Condition 5 is not satisfied.*
- *For Clash Condition 6, we observe that each nominal can only be in the label of one nominal node due to the initialisation of the completion graph, the expansion rules, the definition of V, and due to Condition* C12. *In particular, the latter one*

guarantees that we do not add an expanded node, say v^n, labelled with a nominal that is also in the label of a processed node, say w', since the caching of w' would, by definition of V_p, be invalid and, then, Condition C12 *would have invalidated the caching of v' with $v^n = \mathsf{mergedTo}^{\mathcal{M}^n}(v')$ such that we would not have added v^n. Furthermore, since G' and G^n are not clashed, there also cannot be the cached nodes v' and w' in G' with $v' \dot{\neq}' w'$ which are merged for the expansion, i.e.,* $\mathsf{mergedTo}^{\mathcal{M}^n}(v') = \mathsf{mergedTo}^{\mathcal{M}^n}(w')$*. In particular, Condition* C13 *ensures that the caching would then have been invalidated for v' and w', hence Clash Condition 6 is not satisfied.*

Also, the tableau expansion rules (cf. Table 1) cannot be applied to G. In particular, blocking of nodes in G can be established as in G' and G^n. This can trivially be observed for the processed nodes, i.e., for non-cached nodes from G', since they are only blocked by other nodes from G', i.e., by nodes that are also available in G. Analogously, the expanded nodes can be blocked as in G^n since it is ensured with Condition C11 *that the corresponding blocker nodes are also transferred into G. To be more precise, Condition* C11 *would invalidate the caching of blocked nodes if the caching of the blocker nodes became invalid. Note that if the blocker node does not stem from a cached node and gets pruned because it is no longer connected to a root or nominal node that represents an individual from the knowledge base, then also the blocked node gets pruned since it is a descendant of the blocker node and can also not be connected to such a root or nominal node.*

From Definitions 14 and 15, we can further observe that the tableau expansion rules are not applicable:

- *For the $\sqsubseteq_1$-, $\sqsubseteq_2$-, $\sqcap$-, $\sqcup$-, and* Self*-rules, we can argue as for Clash Conditions 1, 2, and 3. These rules are only applicable on single nodes, but since these rules are neither applicable for the non-cached nodes in G', i.e., the processed nodes in G, nor for the nodes in G^n, i.e., the expanded nodes in G, they are also not applicable for G, otherwise our assumption that G^n is fully expanded and all non-cached nodes in G' are fully expanded would be contradicted. In particular, the data w.r.t. V, E, and $\mathcal{L}$ for these nodes stems either entirely from G' (if a processed node is involved) or from G^n. Hence, the $\sqsubseteq_1$-, $\sqsubseteq_2$-, $\sqcap$-, $\sqcup$-, and* Self*-rules cannot be applicable for G.*
- *For the $\exists$-rule, we observe that it cannot be applicable for processed nodes since the non-cached nodes in G' are fully expanded and the expansion only replaces the neighbouring cached nodes with nodes that have at least the same concepts in their labels. For expanded nodes, we have two cases.*
  First, we consider an expanded node v^n that does not stem from a node in G^d, i.e., there is no $v^d \in V'$ such that $v^n = \mathsf{mergedTo}^{\mathcal{M}^n}(v^d)$. Let us assume that the expanded completion graph G does not contain an r-neighbour node with C in its label for a concept $\exists r.C \in \mathcal{L}(v^n)$. Since the completion graph G^n is fully expanded and all neighbours that do not stem from non-cached nodes have been copied to G, the r-neighbour node w^n of v^n in G^n with C in its label must stem from a node $w^d \in V^d$, i.e., $v^n = \mathsf{mergedTo}^{\mathcal{M}^n}(v^d)$ where $\mathsf{mergedTo}^{\mathcal{M}'}(v^d)$ is not cached. However, Condition C10 *would have invalidated the caching of every ancestor node in G' that is causing the creation of v^n. Hence, v^n would have been pruned for the expansion and, therefore, our assumption is incorrect.*

Let us now consider the other case where the expanded node v^n does stem from a node in G^d, i.e., $v^n = \mathsf{mergedTo}^{\mathcal{M}^n}(v^d)$. Clearly, v^d must be a cached node in V', otherwise we would not have v^n as an expanded node in G. We can now argue analogously to the previous case by using Condition C6. *In particular, if we assume that we have an expanded node v^n that does not have an r-neighbour node with C in its label for a concept $\exists r.C \in \mathcal{L}(v^n)$, then we analogously observe that the missing/incomplete r-neighbour node w^n of v^n in G^n with C in its label must stem from a node $w^d \in V^d$, i.e., $v^n = \mathsf{mergedTo}^{\mathcal{M}^n}(v^d)$ where $\mathsf{mergedTo}^{\mathcal{M}'}(v^d)$ is not cached. However, Condition* C6 *would have invalidated the caching of v^d if it were necessary to add concepts to non-cached node labels to satisfy $\exists r.C$ for v^n. Hence, our assumption is incorrect and, as a result, the $\exists$-rule is not applicable.*

- *The $\forall$-rule is trivially not applicable for processed nodes, otherwise the non-cached nodes would not be fully expanded in G'. Moreover, the $\forall$-rule is also not applicable for an expanded node v^n with $\forall r.C \in \mathcal{L}(v^n)$ since the definition of E guarantees that only the edges and their role labels to and from processed nodes are different for such an expanded node v^n, where Condition* C10 *ensures that v^n stems from a cached node v', i.e., $v^n = \mathsf{mergedTo}^{\mathcal{M}^n}(v')$, (otherwise the caching of ancestors that would cause the generation of v^n in G^n would be identified as invalid), Condition* C9 *ensures that there is no processed r-neighbour node w' of v^n in the expanded completion graph G if w' is not already in G' an r-neighbour of the node v', and Condition* C3 *guarantees that every processed r-neighbour node w' has C in its label (otherwise the caching of v' would be invalid). Note that C can obviously also not be propagated to another expanded node since no additional edges are added or edge labels are extended in G in comparison to G^n between expanded nodes and since G^n is fully expanded.*
- *For the* ch-*rule, we can argue analogously as for the $\forall$-rule by using Condition* C4.
- *For the $\geqslant$-rule, we argue analogously as for the $\exists$-rule. In particular, the $\geqslant$-rule is not applicable for processed nodes since they are fully expanded and since the expansion only adds and/or extends nodes, node labels, and the $\dot{\neq}$ relation. Moreover, for the application on an expanded node v^n, at least one neighbour node must have been removed/changed by the expansion since G^n is fully expanded, i.e., we have a non-cached node $\mathsf{mergedTo}^{\mathcal{M}'}(w^d)$ in G' for which w^d is not expanded as for G^n such that $\mathsf{mergedTo}^{\mathcal{M}^n}(w^d)$ would be an r-neighbour of v^n, would have C in its label, and would be disjoint with the other neighbours (by $\dot{\neq}^n$). In addition, we again observe that v^n must stem from a cached node v', i.e., $v^n = \mathsf{mergedTo}^{\mathcal{M}^n}(v')$, otherwise the Condition* C10 *would have invalidated the caching of every ancestor that would cause the generation of v^n. However, Condition* C7 *ensures that also the caching of v' would have been invalidated if the existence of enough corresponding neighbour nodes cannot be guaranteed for the expansion, i.e., if too many of those nodes get non-cached that are necessary for the satisfaction of the at least cardinality restriction on v^n.*
- *For the $\leqslant$-rule, we can argue analogously to Clash Condition 5. For a processed node, the $\leqslant$-rule is clearly not applicable (otherwise our assumption that the non-cached nodes in G' are fully expanded would be contradicted) and for an expanded node with $\leqslant m\, r.C$ in its label (where only additional r-neighbour connections to processed nodes with the concept C in their label could be problematic), the Condi-*

tions C10 *and* C5 *ensure that the caching of the corresponding nodes would have been invalidated if the cardinality could be problematic and we had to apply the* $\leqslant$*-rule.*

- *For the* o*-rule, we observe analogously to Clash Condition 6 that each nominal can only be in one nominal node and, therefore, the* o*-rule is trivially not applicable.*
- *For the* NN*-rule, we consider two cases. If the concept* $\leqslant m\, r.C$ *is in the label of a processed nominal node* v *and the expansion were to add a new blockable node* w^n *as predecessor of* v *that is* $\mathsf{inv}(r)$*-neighbour of* v *and has* C *in its label, then Condition* C10 *would have invalidated the caching of those predecessors/ancestors that causing the generation of* w^n*. Note that there cannot be a processed node that is blockable and a corresponding predecessor of* v *with* C *in its label since then the* NN*- or the* $\leqslant$*-rule would be applicable for* v*, which contradicts our assumption that the non-cached nodes are fully expanded. For the other case, we assume that the* NN*-rule is applicable for a concept* $\leqslant m\, r.C$ *in the label of an expanded nominal node* v^n *and that the expansion connects a blockable processed node* w *with* C *in its label as a predecessor of* v^n *such that* w *is* r*-neighbour of* v^n*. Since* v^n *must stem from a cached node* v'*, i.e.,* $v^n = \mathsf{mergedTo}^{\mathcal{M}^n}(v')$ *(otherwise* v^n *would not be a direct neighbour of* w*), we observe that our assumption must be wrong since the Condition* C8 *would have invalidated the caching of* v'*. Also note that* v^n *cannot have corresponding expanded nodes as predecessor for which the* NN*-rule could be applicable since* G^n *is fully expanded. Hence, the* NN*-rule is not applicable.*

Since the tableau expansion rules are not applicable for G *and no clash condition is satisfied for* G*,* G *is fully expanded and clash-free.* □

4 Caching Applications and Extensions

In this section, we present additional applications of the completion graph caching technique, which allow for extending the satisfiability caching of node labels such that nominals are supported (Section 4.1) and for reducing the reasoning effort for incremental ABoxes changes (Section 4.2). Furthermore, we describe an extension that allows for caching the completion graph in a representative way (Section 4.3), thus reducing the memory consumption for large knowledge bases with many similar individuals.

4.1 Satisfiability Caching with Nominals

Caching the satisfiability status of labels of (blockable) nodes in completion graphs is an important optimisation technique for tableau-based reasoning systems [5,6]. If one obtains a fully expanded and clash-free completion graph, then the contained node labels can be cached and, if identical labels occur in other completion graphs, then their expansion (i.e., the creation of required successor nodes) is not necessary since their satisfiability has already been proven. Of course, for more expressive DLs that include, for example, inverse roles and cardinality restrictions, we have to consider pairs of node labels as well as the edge labels between these nodes, to be able to ensure that the processing of successor nodes can be blocked (similar to the pairwise blocking condition), i.e., we have to cache a tuple of the form $(\mathcal{L}(v), \mathcal{L}(w), \mathcal{L}(\langle v, w\rangle), \mathcal{L}(\langle w, v\rangle))$. Unfortunately, this kind of satisfiability caching does not work for DLs with nominals. In particular, connections to nominal nodes can be used to propagate new concepts from one blockable node in the completion graph to any other blockable node, whereas for DLs without nominals, the consequences can only be propagated from or to successor nodes. Hence, the caching of node labels is not easily possible and state-of-the-art reasoning systems usually deactivate this kind of caching for knowledge bases with nominals.

However, in combination with completion graph caching, we re-gain the possibility to cache some labels of nodes for knowledge bases with nominals. Roughly speaking, we first identify whether nodes "depend on" nominals as defined next. The labels of such nodes can be cached if all nodes for the dependent nominals, i.e., the nodes those nominals on which the nodes depend, are cached w.r.t. the initial completion graph. As a consequence, we can then block the processing of nodes with identical labels in other completion graphs if the caching of their dependent nominals is not invalid. In order to describe this technique in more detail, we first define nominal dependency in the following.

Definition 16 (Nominal Dependency). *Let $G = (V, E, \mathcal{L}, \dot{\neq}, \mathcal{M})$ be a completion graph, then a node $v \in V$* depends on a nominal $\{a\}$ *if*

- $\{a\} \in \mathcal{L}(v)$,
- *v is blockable and has a successor which depends on $\{a\}$,*
- *v is blockable, there exists a concept $\exists r.C \in \mathcal{L}(v)$ ($\geqslant m\, r.C \in \mathcal{L}(v)$), v has an* $\mathsf{inv}(r)$*-predecessor w with $C \in \mathcal{L}(w)$, v has no r-successor (not m r-successors) with $C \in \mathcal{L}(v)$, and w depends on $\{a\}$, or*
- *v is blocked by w and w depends on $\{a\}$.*

We say that a node v is nominal dependent *if there is a nominal $\{a\}$ in the knowledge base on which v depends and, analogously, we say that v* depends on generated nominals *if there exists a new nominal $\{o\}$ introduced by the* NN*-rule of the tableau algorithm on which v depends.*

Since the tableau algorithm does only construct a successor node for an existential restriction of the form $\exists r.C$ (or a cardinality restriction of the form $\geqslant 1\, r.C$) if there does not already exist a corresponding neighbour node, we need the third condition of Definition 16 to consider the predecessor of a node for the nominal dependency if the predecessor is the only node that satisfies this restriction. The third condition is also required to ensures that the predecessor is considered if the tableau algorithm has merged a successor with the predecessor.

It would also be possible to define nominal dependency by the (sub-)concepts in the label of a node, e.g., we could say that a node v depends on a nominal $\{a\}$ if $\{a\} \in \mathsf{sub}(C)$ and $C \in \mathcal{L}(v)$. Although it is possible to extend this definition to lazy unfolding axioms [1,2], which is, for example, necessary if absorption is used, it is likely that this definition would identify much more nominals as dependent since all non-deterministic alternatives of a concept are considered.

If the tableau algorithm has constructed a fully expanded and clash-free completion graph, we can save the label of a node that is blockable but not blocked together with the label of its blockable predecessor, the labels of the edges between both nodes, and the dependent nominals for both nodes in a (satisfiability) cache as long as both nodes do not depend on newly generated nominals and the nodes for the dependent nominals are cached w.r.t. the initial completion graph. If the labels occur in the same way in another completion graph, then we have to check whether the nodes for the dependent nominals, as stored in the satisfiability cache, are still cached w.r.t. to the initial completion graph. If this is the case, then the processing of the node for which the cache entry matches can be blocked. Of course, if the caching of a node for a dependent nominal becomes invalid, then the processing of the blocked node has to be reactivated. Moreover, if the expansion of a node is blocked since there exists a matching cache entry, we have to use the dependent nominals from the cache entry for this node such that the subsequent caching of other nodes can be correctly continued.

The caching of node labels that depend on newly generated nominal can cause problems. For example, in order to test the satisfiability of a concept $\exists r.\exists r.(A \sqcap \exists s.\{a\})$ for a knowledge base that only contains the axiom $\{a\} \sqsubseteq \leqslant 1\, s^{-}.\top$, we have to initialise a completion graph G with a new node v_0 for which this concept is asserted. We assume that G^d and G^n are cached from the consistency test of the knowledge base, but they are trivial in this example since both contain only the node v_a representing the nominal $\{a\}$, i.e., $\mathcal{L}^d(v_a) = \mathcal{L}^n(v_a) = \{\{a\}, \leqslant 1\, s^{-}.\top\}$. The tableau algorithm can now simply build a fully expanded and clash-free completion graph for testing the satisfiability of $\exists r.\exists r.(A \sqcap \exists s.\{a\})$ by creating an r-successor v_1 of v_0 with $\mathcal{L}(v_1) = \{\exists r.(A \sqcap \exists s.\{a\})\}$ and another r-successor v_2 of v_1 with $\mathcal{L}(v_2) = \{\{o\}, A \sqcap \exists s.\{a\}, A, \exists s.\{a\}\}$, where $\{o\}$ is a new nominal and v_a is an s-neighbour of v_2. Obviously, v_0 and v_1 depend on the nominal $\{o\}$ which does not exist in G^d and, therefore, it cannot be detected whether new consequences are propagated over nominal nodes. To show this with the example, let us assume that we nevertheless create a cache entry for v_1 together with v_0, i.e., $(\{\exists r.(A \sqcap \exists s.\{a\})\}, \{\exists r.\exists r.(A \sqcap \exists s.\{a\})\}, \emptyset, \{r\}, \{\{o\}\})$, where $\{\{o\}\}$ contains the dependent nominals. If we analogously also cache the labels of the first nodes of a completion graph that shows the satisfiability of a concept $\exists r.\exists r.(\neg A \sqcap \exists s.\{a\})$, then a we cannot detect a clash in a completion graph where we tests whether $\exists r.\exists r.(A \sqcap \exists s.\{a\})$ and $\exists r.\exists r.(\neg A \sqcap \exists s.\{a\})$ are simultaneously satisfiable on different individuals. In particular, both successors of both nodes would match the cache entries and, therefore, any further expansion would be blocked. However, a further expansion would force both the concepts $A \sqcap \exists s.\{a\}$ and $\neg A \sqcap \exists s.\{a\}$ into the label of one new nominal node and, therefore, both tested concepts cannot be satisfied simultaneously. Such problems can be simply avoided by caching only labels that do not depend on newly generated nominals. Again, this restriction is more restrictive than necessary, but since many ontologies do not force many newly generated nominals, this is usually not a problematic limitation.

Analogously to more precise blocking conditions [15], it is also possible to refine the satisfiability caching in such a way that only the concepts of one node label have to be saved. However, this would make it necessary to check for all concepts whether there could be some interaction with the predecessor (e.g., whether there exists a concept of the form $\forall r.C$ that could propagate C to the predecessor) and special care require cardinality restrictions for which successors must potentially be merged with predecessors.

4.2 Incremental Reasoning for Changed ABoxes

Many reasoning systems restart reasoning from scratch if a few axioms in the knowledge base have changed. However, it is not very likely that a few changes in the ABox of a knowledge base have a large impact on reasoning. In particular, many ABox assertions only influence the local neighbourhood of the modified individuals and, therefore, the results of reasoning tasks such as classification are often not affected, even if nominals are used in the knowledge base. The presented completion graph caching provides a technique to detect the size of the impact that is caused by changes in the ABox of a knowledge base. This can be used to reduce the reasoning effort for many reasoning tasks. In particular, for the new initial consistency test that is required for the changed ABox assertions, we only have to re-build those parts of the completion graph that are influenced by the changes and cannot be expanded as in the previous completion graph. For higher level reasoning tasks, tracking which nodes of the cached completion graph are potentially used to show the (un-)satisfiability of consistency queries for tested constructs (e.g., subsumption between atomic concepts), allows for identifying whether and which parts of the reasoning tasks have to be repeated. The idea of tracking those parts of a completion graph that are potentially relevant/used for the calculation of higher level reasoning tasks and comparing them with the changed parts in the new completion graph has already been proposed for answering conjunctive queries under incremental ABox updates [11], but the completion graph caching simplifies the realisation of this technique and significantly reduces the overhead for identifying those parts of higher level reasoning tasks that have to be re-computed. Moreover, with the completion graph caching, also very expressive DLs such as $\mathcal{SROIQ}$ can be supported.

In order to describe the approach based on completion graph caching in more detail, we next define modifications for knowledge bases.

Definition 17 (Modification). *Let $\mathcal{K}$ be a knowledge base and $\varDelta^-, \varDelta^+$ disjoint sets of ABox assertions, then the pair $\varDelta = (\varDelta^-, \varDelta^+)$ is a* modification w.r.t. $\mathcal{K}$. Applying *$\varDelta$ to $\mathcal{K}$ yields the knowledge base $\mathcal{K}^{\mp\varDelta} = (\mathcal{K} \setminus \varDelta^-) \cup \varDelta^+$.*

Note, the assumption that $\varDelta^-$ and $\varDelta^+$ are disjoint is without loss of generality. For convenience, we further assume that all ABox assertions are represented as axioms of the form $\{a\} \sqsubseteq C$ such that no adaptation of the tableau expansion rules is required. Furthermore, we assume that we have cached $G^d_{\mathcal{K}}$ ($G^n_{\mathcal{K}}$) as the last deterministic (fully expanded and clash-free) version of the completion graph that shows the satisfiability of $\mathcal{K}$. In the remainder, we also just say that $\varDelta$ is a modification when we mean that $\varDelta$ is a modification w.r.t. $\mathcal{K}$.

Obviously, after applying a modification, we first have to check whether the changes influence the consistency, i.e., whether $\mathcal{K}^{\mp\varDelta}$ is consistent. In order to facilitate the handling of the modifications, we define directly changed individuals and nodes as follows.

Definition 18 (Direct Changes). *Let $\mathcal{K}$ be a knowledge base and $\varDelta = (\varDelta^-, \varDelta^+)$ a modification, then the* set of directly changed individuals w.r.t. *$\varDelta$ is $\{a \mid \{a\} \sqsubseteq C \in \varDelta^- \cup \varDelta^+\}$. Analogously, the* set of directly changed nodes w.r.t. $\varDelta$ and a completion graph $G = (V, E, \mathcal{L}, \dot{\neq}, \mathcal{M})$ for $\mathcal{K}$ *is $\{v \in V \mid \{a\} \in \mathcal{L}(v) \wedge a$ is a directly changed individual$\}$.*

Note, in practice it is usually the case that the sizes of Δ^- and Δ^+ are relatively small compared to the number of assertions that are already in $\mathcal{K}$. This is not a requirement for the incremental reasoning approach based on completion graph caching (in the worst case, we simply have to re-built the entire graph), but it is very beneficial since this usually also means that fewer parts of the completion graph are affected by the changes.

4.2.1 Incremental Consistency Checking for Changed ABoxes Since there could be some assertions in Δ^- that are removed from $\mathcal{K}$, we cannot simply re-use $G^d_{\mathcal{K}}$ or $G^n_{\mathcal{K}}$. In particular, we cannot initialise a new completion graph $G_{\mathcal{K}^{\mp\Delta}}$ for $\mathcal{K}^{\mp\Delta}$ from $G^d_{\mathcal{K}}$ since the consequences of the removed assertions cannot directly be identified. Thus, we have to initialise $G_{\mathcal{K}^{\mp\Delta}}$ from scratch, but we use the same nodes to represent individuals as for $G^d_{\mathcal{K}}$. We then start to deterministically process the directly changed nodes in $G_{\mathcal{K}^{\mp\Delta}}$ and we continue with the processing of neighbours w.r.t. $G_{\mathcal{K}^{\mp\Delta}}$ and $G^d_{\mathcal{K}}$ until the nodes of $G_{\mathcal{K}^{\mp\Delta}}$ are "compatible" to the corresponding nodes in $G^d_{\mathcal{K}}$. Roughly speaking, compatibility ensures that the nodes have the same deterministic consequences as the corresponding nodes in $G^d_{\mathcal{K}}$ and, as a consequence, we can simply "copy" the parts for the remaining nodes from $G^d_{\mathcal{K}}$ as soon as compatibility is achieved. After obtaining the new deterministic completion graph, we can continue the expansion also with non-deterministic rule applications until the nodes are cached w.r.t. $G^n_{\mathcal{K}}$ or fully expanded.

Definition 19 (Compatibility). *Let $G = (V, E, \mathcal{L}, \dot{\neq}, \mathcal{M})$ and $G^d = (V^d, E^d, \mathcal{L}^d, \dot{\neq}^d, \mathcal{M}^d)$ be completion graphs, then a node $v \in V$ is* compatible *with G^d if $v \in V^d$ and $\mathcal{L}(v) = \mathcal{L}^d(v)$.*

Please note that only the nodes for individuals are shared in both completion graphs and, therefore, only such nodes can be compatible. Also note that it is not necessary to consider inequality information for the compatibility since inequalities can only be non-deterministically added to nodes that represent individuals. Likewise, relevant changes through merging are directly represented in the labels (e.g., the merging of nodes for the individuals a and b unites the nominal concepts $\{a\}$ and $\{b\}$ in the same label) and, therefore, it is also not necessary to consider the merging history for compatibility.

In order to achieve compatibility, first the directly changed nodes are deterministically processed and then the deterministic processing is extended step by step to other related nodes. This is continued until enough compatible nodes are obtained such that the deterministic expansion in $G_{\mathcal{K}^{\mp\Delta}}$ of the remaining nodes is possible as in $G^d_{\mathcal{K}}$ without having consequences of removed assertions. In other words, we process those nodes that are related or connected to directly changed nodes without crossing a compatible node, i.e., those nodes are processed that are not "protected" by compatible nodes from the changes. We refer to the nodes that are updated in the deterministic completion graph as "compatibility changed nodes", which are defined as follows.

Definition 20 (Compatibility Changes). *Let $\mathcal{K}$ be a knowledge base, $\Delta = (\Delta^-, \Delta^+)$ a modification, and $\mathcal{K}^{\mp\Delta}$ the result of applying Δ. Furthermore, let $G^d_{\mathcal{K}}$ and $G_{\mathcal{K}^{\mp\Delta}}$ be completion graphs for $\mathcal{K}$ and $\mathcal{K}^{\mp\Delta}$, respectively. We say that a node v is* compatibility changed *if v is not compatible w.r.t. $G^d_{\mathcal{K}}$, and for which*

- *v has a directly or compatibility changed neighbour node w w.r.t. $G^d_{\mathcal{K}}$ or $G_{\mathcal{K}^{\mp\Delta}}$,*
- *there exists a directly or compatibility changed node w and $\mathsf{mergedTo}^{\mathcal{M}}(v) = w$ or $\mathsf{mergedTo}^{\mathcal{M}^d}(v) = w$ or $\mathsf{mergedTo}^{\mathcal{M}}(w) = v$ or $\mathsf{mergedTo}^{\mathcal{M}^d}(w) = v$.*

Note, in order to keep the definition simple, the compatibility changed nodes can be from $G^d_{\mathcal{K}}$ and $G_{\mathcal{K}^{\mp\Delta}}$ and they can also consist of those nodes from these completion graphs that were pruned.

In principle, it would also be possible to deterministically process all nodes in $G_{\mathcal{K}^{\mp\Delta}}$ and then simply check which nodes are not compatible. However, we are obviously interested in processing as few nodes as possible and, therefore, we have to achieve a form of compatibility by processing nodes step by step. This can be realised by selecting the nodes that have to be processed next according to the candidates that could match the conditions for the compatibility changed nodes. Of course, we try to keep the processing local, i.e., we first process the closer neighbourhood of directly changed nodes that are potentially relevant for the compatibility. For this, we can also analyse which consequences of $G^d_{\mathcal{K}}$ are still missing and from which nodes they were propagated in order to determine the node that should be processed next. In addition, we only process role assertions if both individuals are already presented in the completion graph.

If compatibility is achieved, we can build a new deterministic completion graph $G^d_{\mathcal{K}^{\mp\Delta}}$ by combining $G_{\mathcal{K}^{\mp\Delta}}$ and $G^d_{\mathcal{K}}$. In the worst case, compatible nodes do not exist (e.g., if assertion are removed from $\mathcal{K}$ for which consequences are propagated to all nodes in $G^d_{\mathcal{K}}$) and, as a consequence, all nodes have to be processed to obtain $G^d_{\mathcal{K}^{\mp\Delta}}$. For many ontologies, however, the consequences are only propagated to a few neighbour nodes and, therefore, also only a few neighbours have to be processed until compatibility is achieved. Please note that the reasoner is often able to re-derive the consequences for (blockable) nodes very quickly since they often cache how certain node labels are expanded by the tableau algorithm. Hence, the effort for the deterministic re-processing of a few nodes and their blockable successors can usually be neglected.

Combining $G_{\mathcal{K}^{\mp\Delta}}$ and $G^d_{\mathcal{K}}$ to obtain $G^d_{\mathcal{K}^{\mp\Delta}}$ is straightforward, we simply prune the compatibility changed as well as the compatible nodes in $G^d_{\mathcal{K}}$ and add the nodes of $G_{\mathcal{K}^{\mp\Delta}}$. Note that it might be necessary to merge some of the nodes after the combination of both completion graphs, which is, however, not problematic. In particular, we potentially also have to process non-deterministic rule applications for the compatibility changed nodes in order to generate a new fully expanded and clash-free completion graph $G^n_{\mathcal{K}^{\mp\Delta}}$. For this, we can simply continue the processing of $G^d_{\mathcal{K}^{\mp\Delta}}$ by using the completion graph caching for all nodes that are not compatibility changed, i.e., we directly set the caching for the compatibility changed nodes to invalid and continue/reactivate the processing of the remaining nodes as described in Section 3. $G^n_{\mathcal{K}^{\mp\Delta}}$ is then obtained by extending the expanded version of $G^d_{\mathcal{K}^{\mp\Delta}}$ by data for the cached nodes from $G^n_{\mathcal{K}}$. Since our completion graph caching only requires a graph that contains the consequences of a fully expanded and clash-free completion graph, the combination with the data from $G^n_{\mathcal{K}}$ for the cached nodes is simply possible by "copying" those consequences that are potentially related to these nodes. In practice, of course, actual copying is avoided for the majority of the nodes by simply referring to the corresponding nodes in $G^n_{\mathcal{K}}$.

We can now identify the nodes that are also indirectly affected by the ABox changes as follows.

Definition 21 (Indirect Changes). *Let $\mathcal{K}$ be a knowledge base, $\varDelta$ a modification, $G_{\mathcal{K}^{\mp\varDelta}}$ as well as $G^d_{\mathcal{K}}$ be completion graphs as in Definition 20, and $G^n_{\mathcal{K}}$ a completion graph for $\mathcal{K}$ that contains the consequences of a fully expanded and clash-free completion graph. Furthermore, let $G' = (V', E', \mathcal{L}', \dot{\neq}', \mathcal{M}')$ be the completion graph obtained from $G_{\mathcal{K}^{\mp\varDelta}}$ such that each node of G' is*

- *fully expanded, or*
- *not compatibility changed w.r.t. $G^d_{\mathcal{K}}$ and cached w.r.t. $G^d_{\mathcal{K}}$ and $G^n_{\mathcal{K}}$.*

We say that a node v in G' is indirectly changed *if*

- *v is directly or compatibility changed,*
- *v is not cached, or*
- *v is a nominal node, v has a concept $\leqslant m\, r.C$ in its label, and v has a compatibility changed $\mathsf{inv}(r)$-predecessor w with $C \in \mathcal{L}'(w)$.*

Note that the last condition is necessary to detect changes that are relevant w.r.t. the application of the NN-rule on nominal nodes. In particular, if such nominal nodes were not identified as "changed", then we could potentially miss that new consequences are enforced by merging newly generated nominal nodes.

Example 3. Let us assume that we remove the ABox assertion $\{c\} \sqsubseteq \exists s.\{b\}$ from the knowledge base $\mathcal{K}$ of Example 2 and add the axiom $\{c\} \sqsubseteq \exists r.A_4$ instead. Thus,

$$\begin{aligned} \varDelta &= (\{\{c\} \sqsubseteq \exists s.\{b\}\}, \{\{c\} \sqsubseteq \exists r.A_4\}) \\ \text{and } \mathcal{K}^{\mp\varDelta} &= \{\{a\} \sqsubseteq B \sqcup A_1, \{b\} \sqsubseteq B \sqcup \exists r.\exists r.(\{c\} \sqcap A_2), \\ &\qquad \{b\} \sqsubseteq B \sqcup \forall s^-.A_3, \{c\} \sqsubseteq \exists r.A_4\}. \end{aligned}$$

Obviously, only the individual c has been changed directly.

In order to test the consistency of $\mathcal{K}^{\mp\varDelta}$, we initialise a new completion graph $G_{\mathcal{K}^{\mp\varDelta}}$, where we use the same nodes for individuals as for G, i.e., v_a, v_b, and v_c for a, b, and c, respectively. We start the deterministic processing of the directly changed node v_c and continue with the deterministic processing of its neighbour v_b w.r.t. G for which we archive compatibility, i.e., we do not have to process v_a since we know that v_a has the same deterministic consequences. By adding these consequences to $G_{\mathcal{K}^{\mp\varDelta}}$, we obtain $G^d_{\mathcal{K}^{\mp\varDelta}}$, which only differs from G^d by having the concept $\exists r.A_4$ instead of $\exists s.\{b\}$ in the label of v_c. Now, we also process the completion graph non-deterministically and since the caching for v_c is invalid, also the caching of v_b is invalid. Finally, we obtain $G^n_{\mathcal{K}^{\mp\varDelta}}$ by adding the non-deterministic consequences of G^n for v_a and we have v_c and v_b as indirectly changed nodes.

The indirectly changed nodes can now be used for higher level reasoning tasks to determine which parts have to be recalculated, which we describe in more detail next.

4.2.2 Incremental Classification for Changed ABoxes Tableau-based reasoning systems usually classify a knowledge base by performing a satisfiability test for every atomic concept in the knowledge base and by testing the pairwise subsumption relations for all atomic concepts. These tests are reduced to consistency checks and the results can then be used to build the subsumption hierarchy of atomic concepts. If a knowledge base uses nominals, then changes in the ABox can influence the satisfiability of these concepts and also whether the subsumptions hold. Thus, it is in principle necessary to verify all computations for the classification after the ABox has changed.

However, if we track, for all completion graphs generated during classification, which nodes of the cached completion graph are modified, then we can compare them with the indirectly changed nodes from the incremental consistency test of the changed ABox. In other words, we collect for all generated completion graphs the "change dependent nodes" and check whether they are influenced by the changes of the ABox. If a change dependent node is missing in the new deterministic completion graph $G^d_{\mathcal{K}^{\mp\Delta}}$ or there is an overlap with the indirectly changed nodes, then we know that the changes in the ABox could potentially influence a consistency check for the classification and we have to re-compute the corresponding satisfiability and subsumption tests.

Definition 22 (Change Dependency). *Let $G^d = (V^d, E^d, \mathcal{L}^d, \dot{\neq}^d, \mathcal{M}^d)$ be a completion graph with only deterministically derived consequences and $G^n = (V^n, E^n, \mathcal{L}^n, \dot{\neq}^n, \mathcal{M}^n)$ a completion graph that contains the consequences of a fully expanded and clash-free expansion of G^d. Moreover, let G be an extension of G^d and $G' = (V', E', \mathcal{L}', \dot{\neq}', \mathcal{M}')$ a completion graph obtained from G by rule applications. We say that a node $v \in V'$ is* change dependent *if $v \in V^d$ and*

- *v is not cached w.r.t. G^d and G^n, or*
- *there exists $\mathcal{L}'(\langle v, w\rangle)$ ($\mathcal{L}'(\langle w, v\rangle)$) such that there is no $\mathcal{L}^d(\langle v, w\rangle)$ ($\mathcal{L}^d(\langle w, v\rangle)$).*

Note that it is required to analyse and save the change dependent nodes for all generated completion graphs, i.e., also for those that contain clashes.

The technique can obviously be refined by managing for each atomic concept a separate set of change dependent nodes. After the indirectly changed nodes have been determined for a changed ABox, we can compare them with all sets of change dependent nodes and re-compute the satisfiability and subsumptions only for the potentially affected concepts. However, we assume that the the classification result is not often influenced by ABox changes and, therefore, it can also be assumed that this additional overhead does usually not pay off.

4.2.3 Incremental Realisation for Changed ABoxes

The realisation reasoning task is typically realised with instance tests reduced to consistency checks for each individual and atomic concept in a knowledge base. Analogously to classification, we can keep track of the change dependent nodes and re-compute only those instance tests that could be affected by the ABox changes. Since a knowledge base might have many individuals and atomic concepts, the separate tracking of the change dependent nodes for each instance test easily becomes impractical. However, we can simply make an appropriate trade-off. In particular, we can collect the change dependent nodes for the individuals only or we can cluster the individuals by some form of neighbourhood and collect a set of change dependent nodes only once for each cluster. Thus, if the changes in the ABox have only a locally limited influence, then many instance tests do not have to be repeated.

Example 4. The sets of change dependent nodes for the individuals a, b, and c w.r.t. the instance tests of the concept $\exists r.\top$ in Example 2 are $\{v_a, v_b\}$, $\{v_b\}$, and $\{v_c, v_b\}$, respectively. If the knowledge base $\mathcal{K}$ of Example 2 is modified by $\varDelta = (\{\{c\} \sqsubseteq \exists s.\{b\}\}, \{\{c\} \sqsubseteq \exists r.A_4\})$, then we obtain v_b and v_c as indirectly changed nodes (cf. Example 3). Thus, we have to recalculate the instance tests for all individuals and, as a result, we obtain that now also c is an instance of $\exists r.\top$. In contrast, the sets of change dependent nodes for the individuals a, b, and c w.r.t. the instance tests of the concept $\exists s.\top$ are $\{v_a\}$, $\{v_b\}$, and $\{v_c, v_b\}$, respectively. Hence, after the ABox changes, we only have to re-compute the instance tests for b and c and they would reveal that c is no longer an instance of $\exists s.\top$.

4.3 Representative Caching

Clearly, caching an entire completion graph can require a lot of memory, especially if an ABox contains many individuals and the tableau algorithm has to generate many successors for these individuals for the construction of a completion graph. Hence, if the ABox is very large, then it easily becomes impractical to handle such knowledge bases with state-of-the-art tableau-based reasoners on standard computer systems. There exist several approaches that try to ease the work for reasoners by reducing the size of the ABox (e.g., by summarisation and refinement [4], abstraction [7], or modularisation [30]), but for more expressive DLs with non-determinism (e.g., $\mathcal{SROIQ}$), it is often not possible to completely avoid the handling of such ABoxes with fully-fledged and highly optimised reasoning systems, which are usually based on tableau algorithms. However, tableau-based reasoning systems are usually not able to dynamically store and load parts of completion graphs to and from secondary memory storages (e.g., hard drives) and, therefore, the size of the ABox that can be handled by such reasoners is limited by the main memory. Thus, knowledge bases with large ABoxes that use expressive language features often cause problems in practise and can potentially not be handled satisfactorily.

In the following, we adapt the presented completion graph caching such that relevant data can be stored in a representative way, which allows for building "local" completion graphs for small subsets of the entire ABox until the existence of a complete completion graph considering all individuals can be guaranteed. To be more precise, if a fully expanded and clash-free completion graph is constructed for a subset of the ABox (e.g., a subset of all individuals and their assertions), then we extract and generalise information from the processed individuals and store them in a representative cache. If we then try to build a completion graph for another subset of the ABox that has some overlapping with a previously handled subset (e.g., role assertions for which edges to previous handled individuals have to be created), then we load the available data from the cache and continue the processing of the overlapping part until it is "compatible", i.e., the expansion of the remaining individuals in the cache can be guaranteed as in the previously constructed completion graphs. Of course, this only works well for knowledge bases for which there is not too much non-deterministic interaction between the separately handled ABox parts. Moreover, compared to the ordinary completion graph caching, we are clearly trading less memory consumption against an increased runtime since more work potentially has to be repeated to establish compatibility. In practice, it is obviously also possible to combine both techniques, i.e., we can use the representative caching only for those individuals of an ontology that can be primarily deterministically handled and, for the remaining individuals, we can build an ordinary completion graph and use the caching as presented in Section 3.

In the following, we first define the "representative cache" which is used to store the derived consequences of already processed parts/individuals of the ABox.

Definition 23 (Representative Cache). *Let $\mathcal{K}$ be a knowledge base and* $\mathsf{fclos}(\mathcal{K})$*,* $\mathsf{Rols}(\mathcal{K})$*, and* $\mathsf{Inds}(\mathcal{K})$ *the set of concepts, roles, and individuals that can occur in $\mathcal{K}$ or in a completion graph for $\mathcal{K}$, respectively. The* representative cache $\mathcal{H}$ w.r.t. $\mathcal{K}$ *is a tuple of the form $(I, C^k, C^p, S^k, D^k, R^p, N^p)$, where*

- $I \subseteq \mathsf{Inds}(\mathcal{K})$ *denotes the set of those individuals that are represented in the cache,*
- $C^k : \mathsf{Inds}(\mathcal{K}) \to 2^{\mathsf{fclos}(\mathcal{K})}$ *is the mapping of individuals to sets of known concepts,*
- $C^p : \mathsf{Inds}(\mathcal{K}) \to 2^{\mathsf{fclos}(\mathcal{K})}$ *denotes the mapping of individuals to possibly instantiated concepts,*
- $S^k : \mathsf{Inds}(\mathcal{K}) \to 2^{\mathsf{Inds}(\mathcal{K})}$ *is the mapping of individuals to known same individuals,*
- $D^k : \mathsf{Inds}(\mathcal{K}) \to 2^{\mathsf{Inds}(\mathcal{K})}$ *denotes the mapping of individuals to known disjoint individuals,*
- $R^p : \mathsf{Inds}(\mathcal{K}) \times \mathsf{Rols}(\mathcal{K}) \to \mathbf{N}_0$ *denotes the mapping of individual role pairs to the number of potentially existing neighbour nodes, and*
- $N^p : \mathsf{Inds}(\mathcal{K}) \to 2^{\mathsf{Inds}(\mathcal{K})}$ *is the mapping of individuals to those individuals that are (potentially indirectly) connected via nominals.*

We say that an individual $a \in I$ is related *to an individual $b \in \mathsf{Inds}(\mathcal{K})$ w.r.t. $\mathcal{H}$ if*

- $\{a\} \sqsubseteq \exists r.\{b\} \in \mathcal{K}$ *or* $\{b\} \sqsubseteq \exists r.\{a\} \in \mathcal{K}$*;*
- $\{a\} \in C^p(b)$ *or* $\neg\{a\} \in C^p(b)$*;*
- $a \in S^k(b)$ *or* $a \in D^k(b)$*; or*
- $a \in N^p(b)$ *or* $b \in N^p(a)$*.*

The known and possible information represented in the cache obviously correspond to the deterministically and non-deterministically derived facts in completion graphs, respectively. Hence, known consequences from the cache can be used to speed up the re-construction of already processed and cached parts in a completion graph since the corresponding facts can deterministically be added to initialise the corresponding nodes, wherefore we do not have to apply rules to re-derive this information. The possible information is in addition required since we have to check whether the re-constructed parts are compatible with the cache or whether we have to further expand related individuals.

Please note that we only store some "generalised" information in the cache, otherwise the data can often not be stored in a representative way. For example, each node label for an individual contains the nominal concept (e.g., $\{a\}$) and the role assertion concepts (e.g., $\exists r.\{b\}$) for such an individual, which is usually very specific for such an individual and not shared by many other nodes. Since such data can, however, easily be re-constructed with the axioms of the knowledge base, we remove them from the labels before writing them to the cache. In the best case, such a "generalised" label is then shared by many individuals and the representative storing significantly reduces the memory consumption. For the same reason, we also use separated mappings for the same and disjoint individuals in the cache although this information is usually also represented within the node labels of a completion graph. In particular, the merging of nodes for individuals unites the nominal concepts for both individuals in the same label. For simplicity, however, we only separate the same and disjoint individuals that can be deterministically derived, i.e., if we non-deterministically add the nominal concept $\{b\}$ to a label of a node v_a that represents the individual a, then b and a are possibly the same individuals, but we map $C^p(a)$ to $\mathcal{L}(v_a) \setminus \{\{a\}\}$, $S^k(a)$ to $\{a\}$, $C^p(b)$ to $\mathcal{L}(v_a)$, and $S^k(b)$ to $\emptyset$, i.e., we only remove the deterministic same/disjoint individual information from the perspective of the individual that is cached. Of course, the stored sets of concepts still contain $\{b\}$ and, thus, these sets can probably not be re-used by many other cache entries. However, we assume that the knowledge base can mostly be handled deterministically and, therefore, this should not be a significant restriction. Besides that, the cache could also simply be extended with mappings that handle "possibly same" and "possibly disjoint" individuals.

The mapping R^p is required to check whether additional neighbour nodes potentially violate cardinality restrictions for a specific role and N^p is used to remember connections (possibly over blockable nodes via nominals) between individuals that are not directly caused by role assertions. Of course, the storage of this information in the cache can also be refined and it depends on the ontology whether and how useful such a refinement is. Note that we only have to store the number of neighbour nodes if there is a problematic at-most cardinality restriction for an individual.

The extraction of required information for the representative caching from the generated completion graphs is straightforward. Non-deterministically derived consequences can usually be distinguished with the branching tags that are associated with all facts in order to support backjumping [1,28]. The indirectly connected individuals can be determined by analysing whether edges to neighbour nodes are labelled with roles that are not direct consequences of role assertions and by tracking the nominal dependency for all blockable nodes (cf. Definition 16). In order to describe the interaction between a representative cache $\mathcal{H} = (I, C^k, C^p, S^k, D^k, R^p, N^p)$ and a completion graph $G = (V, E, \mathcal{L}, \dot{\neq}, \mathcal{M})$ for a knowledge base $\mathcal{K}$ in more detail, we use several auxiliary functions that are defined as follows. For a node $v_a \in V$ that represents an individual $a \in \mathsf{Inds}(\mathcal{K})$ in G, i.e., $\{a\} \in \mathcal{L}(v_a)$, let

- $\mathcal{L}^{\mathsf{det}}(v_a) = \{C \in \mathcal{L}(v_a) \mid C$ is deterministically added to $\mathcal{L}(v_a)\}$, i.e., the function that returns that subset of the label of v_a for which all concepts are deterministically derived;
- $\mathsf{sameInds}(v_a) = \{b \mid \{b\} \in \mathcal{L}^{\mathsf{det}}(v_a) \wedge b \in \mathsf{Inds}(\mathcal{K})\}$, i.e., the function that returns the set of individuals represented by this node;
- $\mathsf{disjInds}(v_a) = \{b \mid \neg\{b\} \in \mathcal{L}^{\mathsf{det}}(v) \vee v_a \dot{\neq} v_b$ is deterministically added and $\{b\} \in \mathcal{L}(v_b)\}$, i.e., the function that returns the set of disjoint individuals;
- $\mathsf{rassCons}(v_a) = \{\exists r.\{b\} \mid \{c\} \sqsubseteq \exists r.\{b\} \in \mathcal{K} \wedge c \in \mathsf{sameInds}(v_a)\}$, i.e., the function that returns the set of role assertion concepts that are related to v_a;

- $\mathsf{possCons}$ be the function that returns the set of generalised concepts that are possibly instantiated by v_a, i.e.,

$$\begin{aligned}\mathsf{possCons}(v_a) = (&\mathcal{L}(v_a) \cup \{\neg\{b\} \mid v_a \dot{\neq} v_b \wedge \{b\} \in \mathcal{L}(v_b) \wedge b \in \mathsf{Inds}(\mathcal{K})\}) \setminus \\ &(\mathsf{rassCons}(v_a) \cup \{\neg\{b\} \mid b \in \mathsf{disjInds}(v_a)\} \cup \{\{b\} \mid b \in \mathsf{sameInds}(v_a)\});\end{aligned}$$

- $\mathsf{knownCons}(v_a) = \mathsf{possCons}(v_a) \cap \mathcal{L}^{\mathsf{det}}(v_a)$, i.e., the function that returns the set of general concepts that are deterministically derived for v_a, i.e., for which it is known that they are indeed instantiated by v_a;
- $\mathsf{usedCard}(v_a, r) = \#\mathsf{mneighbs}^G(v, r, C)$, i.e., the function that returns the number of v_a's r-neighbour nodes; and
- $\mathsf{nocoInds}(v_a) = \{b \mid \{b\} \in \mathcal{L}(v_b) \wedge v_b$ has a blockable successor that depends on $\{a\}\}$ $\cup \{b \mid \{b\} \in \mathcal{L}(v_b) \wedge v_b$ is an r-neighbour of $v_a \wedge$ there is no $\exists s.\{b\} \in \mathsf{rassCons}(v_a)$ for all s with $s \sqsubseteq^* r\}\}$, i.e., the function that returns the set of individuals that are (possibly indirectly) connected to v_a by using nominals.

Of course, we initially have an empty representative cache. In order to start the consistency checking process, we select a subset of the individuals of the knowledge base, say $a_i, \ldots, a_k$, and build a fully expanded and clash-free completion graph for these individuals and their associated ABox assertions of the form $\{a_i\} \sqsubseteq C$ for $1 \leq i \leq k$. Due to role assertions and nominals, it is possible that the completion graph also refers to individuals that are not in the selected subset. The information for all processed or referred individuals must then be stored in the cache. In particular, if a node v_a represents an individual a such that $\{a\} \in \mathcal{L}^{\mathsf{det}}(v_a)$, then we add a to I and set $C^k(a)$ to $\mathsf{knownCons}(v_a)$, $C^p(a)$ to $\mathsf{possCons}(v_a)$, $S^k(a)$ to $\mathsf{sameInds}(v_a)$, $D^k(a)$ to $\mathsf{disjInds}(v_a)$, $N^p(a)$ to $\mathsf{nocoInds}(v_a)$, and $R^p(a, r)$ to $\mathsf{usedCard}(v_a, r)$ for every (possibly inverse) role r in the knowledge base. If $\{a\} \notin \mathcal{L}^{\mathsf{det}}(v_a)$ but $\{a\} \in \mathcal{L}(v_a)$, then we have to interpret all information non-deterministically, i.e., we add a to I and set $C^k(a)$ to $\{\top\}$, $C^p(a)$ to $\mathcal{L}(v_a)$, $S^k(a)$ to $\emptyset$, $D^k(a)$ to $\emptyset$, $N^p(a)$ to $\mathsf{nocoInds}(v_a)$, and $R^p(a, r)$ to $\mathsf{usedCard}(v_a, r)$ for every (possibly inverse) role r in the knowledge base. For simplicity, we keep the information in the cache symmetric, i.e., we also extend $D^k(a)$ to b if we add a to $D^k(b)$. Please note that also information about not selected individuals has to be cached.

Now, let us assume that the representative cache contains the information about the previously processed individuals $a_1, \ldots, a_k$. We continue by selecting the next subset of individuals and their ABox assertions, say $b_1, \ldots b_l$, and by building a completion graph for these individuals and assertions. In case we refer to a cached individual a_i, e.g., by adding the nominal $\{a_i\}$ to a node label, we can load the deterministically derived consequences for a_i from the cache and add them also to the label. Note that the role assertion concepts are not automatically added in order to avoid the repeated processing of all cached individuals in the new completion graph. However, for the overlapping individuals, i.e., the individuals that are represented in the cache and are also used/referred to in the completion graph, we have to establish "compatibility" such that the expansion of the completion graph to the remaining individuals in the cache can be guaranteed as in the previously constructed completion graphs. Analogously to the caching criteria in Definition 14, we define conditions that identify nodes as potentially incompatible with the data of the representative cache, for which we then know that a further expansion of the completion graph is required. In particular, we have to extend the completion graph by related individuals of incompatible nodes w.r.t. the representative cache until the remaining individuals in the cache are protected by a border of compatible nodes. By instantiating related individuals in the completion graph, we also add the corresponding role assertions such that all individuals in the completion graph are correctly connected. Please note that, in contrast to the ordinary completion graph caching, we do not block the processing of nodes, i.e., we fully process all nodes in the completion graph (besides the role assertions to individuals that are not represented in the completion graph) and, if it is necessary, then we add and process nodes also for previously handled individuals such that incompatibility can be resolved.

Definition 24 (Representative Caching). *Let $G = (V, E, \mathcal{L}, \dot{\neq}, \mathcal{M})$ be a completion graph for a knowledge base $\mathcal{K}$ and $\mathcal{H} = (I, C^k, C^p, S^k, D^k, R^p, N^p)$ the representative cache. Furthermore, let $v_a \in V$ be a node that represents an individual $a \in \mathsf{Inds}(\mathcal{K})$ in G, i.e., $\{a\} \in \mathcal{L}(v_a)$. We say that v_a is* incompatible *w.r.t. $\mathcal{H}$ if*

R1 $a \notin I(a)$;
R2 $\mathsf{knownCons}(v_a) \not\subseteq C^k(a)$;
R3 $\mathsf{possCons}(v_a) \not\subseteq C^p(a)$;
R4 $\mathsf{sameInds}(v_a) \not\subseteq S^k(a)$;
R5 $\mathsf{disjInds}(v_a) \not\subseteq D^k(a)$;
R6 $\leqslant m\, r.C \in C^p(a)$, *$v_a$ has n incompatible r-neighbours, $R^p(a, r) + n > m$, and there exists a related individual b of a w.r.t. $\mathcal{H}$ such that there is no node $v_b \in V$ with $\{b\} \in \mathcal{L}(v_b)$ or* $\mathsf{possCons}(v_b) \neq C^p(b)$;
R7 $\mathsf{possCons}(v_a) \neq C^p(a)$ *and v_a has an incompatible neighbour node v_b with $\{b\} \in \mathcal{L}(v_b)$ and $b \in \mathsf{Inds}(\mathcal{K})$;*
R8 $\mathsf{possCons}(v_a) \neq C^p(a)$ *and v_a has a predecessor node that is blockable or contains a newly generated nominal;*
R9 $\mathsf{possCons}(v_a) \neq C^p(a)$, *there exists a node $v_b \in V$ with $\{b\} \in \mathcal{L}(v_b)$ such that $a \in N^p(b)$, and v_b is incompatible or* $\mathsf{possCons}(v_b) \neq C^p(b)$;

Although there does not exist an exact correspondence, the conditions for the representative caching are in principle very similar to the criteria for the ordinary completion graph caching. However, since we only cache data for nodes that represent individuals now, the new conditions are often less accurate and, as a consequence, it is often necessary to re-construct more parts of already processed individuals. In particular, if there is a cached at-most restriction for an individual and not all of the previously constructed neighbour nodes are in the completion graph such that they can be correctly counted, then we can only analyse whether the number of potential new neighbour nodes together with the previous neighbours could violate the at-most restriction (cf. Condition R6). If all related individuals of a node with a cached at-most restriction are represented in the completion graph and all these nodes are expanded as in the cache, then we know that the tableau algorithm considers all of the previously constructed neighbours and, in case it can build a fully expanded and clash-free completion graph, then the cardinality restriction is also compatible with the representative cache. Condition R7 is used to enforce that propagations of non-deterministic consequences from cached neighbour nodes are considered. Analogously, Conditions R8 and R9 ensure that nodes are expanded as in the cache if there are indirect connections via nominals over blockable nodes such that the propagation of new consequences over these blockable nodes can be excluded.

As already mentioned, if there is an incompatible node in the completion graph, then we check in the representative cache whether there are related individuals that also have to be instantiated and processed. Again, we can analyse the cached concepts of related individuals in order to prioritise the handling of those individuals in the completion graph for which it is likely that they allow for establishing compatibility for many other nodes as soon as possible. For example, if we build a completion graph for a newly selected and previously not processed individual b for which a neighbour node v_a exists that represents a cached individual a and v_a is incompatible w.r.t. Condition R7 due to a missing concept A in it's label, then we can prioritise the instantiation and processing of a related neighbour individual c for which a (non-deterministically derived) concept of the form $\forall r.A$ is associated. If it is possible to establish the compatibility of v_a in this way, then we potentially do not have to instantiate and process many other related individuals of a. In principle, we can also try to non-deterministically re-use cached non-deterministic information, e.g., by non-deterministically adding A for v_a. If we can still find a fully expanded and clash-free completion graph with this non-deterministically added information, then we can often establish compatibility without repeatedly processing many previously cached individuals. In the worst case, of course, one has to consider and re-process all individuals that contribute some non-deterministic consequences to clashes until a fully expanded and clash-free completion graph can be found.

After the construction of the completion graph for the newly selected individuals, we have to update the data in the cache. In principle, this is straightforward: for all incompatible nodes, we can simply replace the data in the cache with the (potentially new) information from the completion graph, and for the compatible nodes that represent individuals, we have to update R^p and N^p by the additional neighbour nodes and the new individuals that are indirectly connected via nominals over blockable nodes, respectively.

Example 5. The completion graph constructed for Example 2 can be cached with $\mathcal{H} = (I, C^k, C^p, S^k, D^k, R^p, N^p)$ by adding a, b, c to I, by mapping

- $C^k(a)$ to $\{\top, B \sqcup A_1\}$, $C^p(a)$ to $\{\top, B \sqcup A_1, A_1\}$, $S^k(a)$ to $\{a\}$,
- $C^k(b)$ to $\{\top, B \sqcup \exists r.\exists r.(\{c\} \sqcap A_2), B \sqcup \forall s^-.A_3\}$, $C^p(b)$ to $\{\top, B \sqcup \exists r.\exists r.(\{c\} \sqcap A_2), B \sqcup \forall s^-.A_3, \exists r.\exists r.(\{c\} \sqcap A_2)\}$, $S^k(b)$ to $\{b\}$,
- $C^k(c)$ to $\{\top\}$, $C^p(c)$ to $\{\top, \{c\} \sqcap A_2, A_2, A_3\}$, $S^k(c)$ to $\{c\}$, $N^p(c)$ to $\{a\}$, and

by setting all other mappings for a, b, c to $\emptyset$ (since we do not have at-most cardinality restrictions, R^p is not relevant).

Let us now assume that we have an additional individual d with $\{d\} \sqsubseteq \exists s.\{a\}$ in the knowledge base and we select d for building the next "local" completion graph compatible with $\mathcal{H}$ such that the overall consistency of the knowledge base can be shown. Since we have the role assertion concept $\exists s.\{a\}$ for d, we extend the completion graph consisting of node v_d with $\mathcal{L}(v_d) = \{\top, \{d\}, \exists s.\{a\}\}$ by a node v_a whose label is initialised from the cache, i.e., $\mathcal{L}(v_a) = \{\top, \{v_a\}, B \sqcup A_1\}$. By completing the processing for v_a by choosing the disjunct A_1, we obtain a fully expanded and clash-free completion graph for which compatibility with $\mathcal{H}$ is achieved. $\mathcal{H}$ can now be updated by adding d to I and by mapping $C^k(d)$ and $C^p(d)$ both to $\{\top\}$.

If we also have another individual e with $\{e\} \sqsubseteq B \sqcup \exists s.(\{c\} \sqcap A_4)$ and we choose the non-deterministic alternative $\exists s.(\{c\} \sqcap A_4)$ for the node that represents e, then we also have to initialise and process a node v_c for c. Obviously, c is incompatible due to Condition R3 and, therefore, we have to extend the completion graph by a node for b that is related to c by $N^p(c)$ and by the role assertion concept $\exists s.\{b\}$. Again, we can achieve compatibility by processing the node for b as in the original completion graph. Note that Condition R9 does not apply since the label of the node for b still matches $C^p(b)$.

5 Related Work

Especially optimisations for the instance checking problem are closely related to the presented caching approach. For more expressive DLs, it is often required to systematically try to build (counter-)models in order to decide whether an individual is an instance of a concept. Of course, there are many optimisations, such as summarisation [4], abstraction and refinement [7], bulk processing and binary retrieval [9], (pseudo) model merging [10], extraction of known/possible instances from model abstractions [20], which try to keep the number of cases, where (counter-)models have to be constructed, as small as possible, however, the model construction can, in general, not be completely avoided. Since tableau algorithms are dominantly used for reasoning with expressive DLs, it is necessary to have optimisations that improve their handling for such instance tests which are usually also reduced to consistency checks. Although such optimisations have been presented and realised, e.g., by extending absorption to ABoxes [31] or by partitioning the ABox into small islands [30], there is no approach that can be generally applied for $\mathcal{SROIQ}$, the DL underlying OWL 2.

In particular, it is not clear how to extend the partitioning approach such that instance checking w.r.t. arbitrary concepts is supported. So far, the island partitions are statically calculated upfront and then only that island is used for further calculation to which the individual from the query/instance test belongs to. In order to be able to restrict the calculations to such an island, all interactions with other islands have to be excluded, which is, however, not easily possible if arbitrary concepts must be handled. For example, by querying for instances of concepts of the form $\exists r. \ldots \exists s.C$, where r, s are possibly inverse or even complex roles, we can enforce that all connected individuals have to be considered. Moreover, for DLs that include the universal role U or nominals, we can query for instances of concepts of the form $\forall r.(\{a\} \sqcap C)$ and $\exists U.C$, which could even require the consideration of all individuals in the knowledge base. Hence, a dynamic partitioning approach would be required, which can, however, not easily and probably also not efficiently be realised. Moreover, it is not clear how one can determine the island partitions upfront if the knowledge bases use very expressive language features such as nominals. For instance, if there is an axiom of the form $\top \sqsubseteq \{a\} \sqcup A$, then it can be necessary to merge the node that represents a with any other node that does not satisfy A in a completion graph and, thus, a cannot simply be split into a separate island before the calculations are completed, i.e., not before we exactly know which nodes must potentially be merged.

The ABox absorption approach extends ABox assertions by “guards” represented as atomic auxiliary concepts in order to construct only those parts of the ABox in completion graphs for which the guards are triggered. For this, the axioms/concepts of the knowledge base are normalised, extended, and absorbed in such a way that the tableau algorithm automatically triggers, while processing these axioms/concept, those guards (i.e., adds the corresponding atomic auxiliary concepts) that ensure the expansion to other individuals in the ABox, wherewith then possible interactions can be detected. The approach has been presented for the DL $\mathcal{SHIQ}$, but it is not clear how it can be extended to $\mathcal{SROIQ}$. In addition, the adaptations w.r.t. normalisation and absorption can significantly influence other optimisations and reasoning task. For example, the classification of consistent $\mathcal{SHIQ}$ knowledge bases can be performed independently from the ABox, but the normalisation and absorption of the axioms in the knowledge bases can introduce additional non-determinism, which can have a significant impact on the performance of satisfiable and subsumption tests. Furthermore, this approach requires that it is analysed upfront, before any reasoning, for which concepts the guards have to be triggered. This is obviously much more vague as a technique that directly uses reasoning data, such as the presented completion graph caching, and, as a consequence, potentially more individuals have to be considered to check whether an individual is an instance of a concept.

Of course, also incremental reasoning has already been considered in several related works for which we briefly point out the differences and similarities to our approach in the following. The presented incremental consistency checking for changed ABoxes is based on the observation that re-building parts of a completion graph is potentially cheaper than tracing exactly on which ABox assertions a derived facts depends. The facts that depend on a removed ABox assertion can then be deleted from the existing graph and, then, the tableau rules can be applied again to obtain a new clash-free and fully expanded completion graph. Although such tracing techniques can be realised also for more expressive DLs (such as $\mathcal{SHIQ}$ and $\mathcal{SHOQ}$) [12], it requires a significant adaptation of expansion rules and increases reasoning time as well as memory consumption. Moreover, non-determinism causes several technical difficulties for the tracing approach. For instance, if removed ABox assertions were involved in the creation of clashes for the consistency test of the original ABox, then it could be necessary to re-evaluate the corresponding non-deterministic alternatives to find a new clash-free and fully expanded completion graph after new ABox assertions are added. Hence, a significant overhead is required to manage the potential impact of such assertions. In contrast, the idea of re-constructing parts from scratch after changes (until “compatibility” is achieved) is already successfully deployed for less expressive DLs [19] and the presented completion graph caching allows for using this approach also for very expressive DLs such as $\mathcal{SROIQ}$.

In some cases it is also possible to check with a syntactical analysis of the knowledge base whether ABox changes could cause new clashes. If this can be excluded, then the construction of a new/updated completion can in principle be spared. For example, if a role assertion of the form $r(a, b)$ is added to a $\mathcal{SHIQ}$ knowledge base, then we know that consistency is trivially preserved if there exist no universal or cardinality restriction in the knowledge base that can have an interaction with the added role instantiation [32]. Of course, especially for knowledge bases that intensively use language features of more expressive DLs, this approach cannot completely avoid the re-computation of the consistency since the syntactic analysis must often be overcautious.

As presented, the completion graph caching also allows for simply detecting whether parts of higher level reasoning tasks have to be re-computed for changed ABoxes by comparing the impact of the changes in the initial completion graph with those parts that were relevant for the calculation of the higher level reasoning tasks. In principle, the same approach has also been proposed for query answering with incrementally changed ABoxes of $\mathcal{SHI}$ knowledge bases [11]. However, without the completion graph caching (or a similar technique), it is required to extend the generated completion graph to an overestimation considering all non-deterministic alternatives and to extract and store a substantial amount of information. Although many improvements are achieved for the evaluated scenarios, the required overhead for deciding whether (parts of) queries have to be re-evaluated can still be substantial. Please note that we have only discussed reasoning improvements for classification and realisation of incremental changed ABoxes with the completion graph caching, but improvements for other higher level reasoning tasks (such as query answering) are analogously possible.

Last but not least, we briefly compare the presented incremental reasoning based on completion graph caching with modularisation-based approaches [3], which compute, for each axiom, the subset of the knowledge base, called module, that is “relevant” to determine whether certain entailments hold. Clearly, an entailment must only be recomputed if an axiom of its module has been changed. Although this allows for handling all types of changes in a knowledge base (i.e., also arbitrarily added or removed GCIs), it is often a very difficult task to calculate precise modules and, therefore, it is often not very suitable for very expressive DLs. In particular, the overhead of calculating and managing the modules can easily be more expensive than a re-computation from scratch. Moreover, in order to calculate the modules, it is, analogously to partitioning approaches, required to know upfront which types of entailments must be considered (e.g., subsumptions between atomic concepts or instances of atomic concepts), which can be problematic for some reasoning tasks such as (conjunctive) query answering.

6 Implementation and Evaluation

The presented completion graph caching is integrated in our reasoning system Konclude [26], which is a tableau based reasoner for the DL $\mathcal{SROIQV}$, i.e., $\mathcal{SROIQ}$ extended by nominal schemas. Besides many state-of-the-art optimisations, such as lazy unfolding, absorption, dependency directed backtracking, satisfiability caching, etc., Konclude also incorporates a saturation procedure to assist the tableau algorithm, whereby a large amount of relatively simple ontologies/simple parts of ontologies can be handled very easily. However, for ontologies that use (non-deterministic) language features of more expressive DLs such as disjunctions and nominals, the saturation easily becomes incomplete and it is required to perform consistency/satisfiability tests with the tableau

Table 2. Ontology metrics for selected benchmark ontologies (A stands for Axioms, C for Classes, P for Properties, I for Individuals, CA for Class Assertions, OPA for Object Property Assertions, and DPA for Data Property Assertions)

Ontology	Expressivity	#A	#C	#P	#I	#CA	#OPA	#DPA
OGSF	$\mathcal{SROIQ}(\mathcal{D})$	1,235	386	179	57	45	58	20
Wine	$\mathcal{SHOIN}(\mathcal{D})$	1,546	214	31	367	409	492	2
DOLCE	$\mathcal{SHOIN}$	1,667	209	317	42	101	36	0
OBI	$\mathcal{SROIQ}(\mathcal{D})$	28,770	3,549	152	161	273	19	1
USDA-5	$\mathcal{ALCIF}(\mathcal{D})$	1,401	30	147	1,214	1,214	12	0
COSMO	$\mathcal{SHOIN}(\mathcal{D})$	29,655	7,790	941	7,817	8,675	3,240	665
DPC1	$\mathcal{ALCIF}(\mathcal{D})$	55,020	1,920	94	28,023	15,445	39,453	0
Oly	$\mathcal{ALCIF}(\mathcal{D})$	35,988	1,223	94	18,360	10,161	25,705	0
UOBM-1	$\mathcal{SHOIN}(\mathcal{D})$	260,728	69	44	25,453	46,403	143,549	70,628
CobCav92	$\mathcal{SROIF}(\mathcal{D})$	659,653	719	110	128,888	419,871	273,620	624
LUBM-1	$\mathcal{ALEHI}^{+}(\mathcal{D})$	100,636	43	32	17,174	18,128	49,336	33,079

algorithm in order to fulfil reasoning tasks (e.g., classification, realisation). Hence, we selected a range of ontologies (cf. in Table 2) for our evaluation, for which indeed some significant processing with the tableau algorithm is required, i.e., ontologies with large and/or non-trivial ABoxes that also use non-deterministic language features. Well-known examples of such ontologies are Wine, DOLCE,[3] OBI,[4] and UOBM [21], but our evaluation dataset also comprises DPC1, Oly, and USDA-5, which are benchmark ontologies about digital cameras and food items that have been used for the evaluation of the ABox absorption technique [31], as well as the Ontology for Genetic Susceptibility Factor (OGSF),[5] the Common Semantic Model[6] (COSMO) ontology, and the relatively large, biological Coburn_Cavender_1992[7] (CobCav92) ontology from the phenoscape project. In addition, we use the well-known LUBM-1 [8] ontology to demonstrate effects for deterministic ontologies.

The evaluation was carried out on a Dell PowerEdge R420 server running with two Intel Xeon E5-2440 hexa core processors at 2.4 GHz with Hyper-Threading and 144 GB RAM under a 64bit Ubuntu 12.04.2 LTS. In order to make the evaluation independent of the number of CPU cores, we used only one worker thread for Konclude. We used 5 minutes as time limit and ignored the time spent for parsing the ontologies as well as writing the results.

Table 3 shows the reasoning times for consistency checking (including preprocessing), classification, and realisation (in seconds) with different completion graph caching techniques integrated in Konclude. Please note that the class hierarchy is required to realise an ontology, i.e., classification is a prerequisite of realisation, and, analogously,

[3] `http://www.loa.istc.cnr.it/old/DOLCE.html`
[4] `http://obi-ontology.org/`
[5] `https://code.google.com/p/ogsf/`
[6] `http://ontolog.cim3.net/cgi-bin/wiki.pl?COSMO`
[7] `https://github.com/phenoscape/phenoscape-data/blob/master/Curation%20Files/completed-phenex-files/Cypriniformes/Coburn_Cavender_1992.xml`

Table 3. Reasoning times for different completion graph caching techniques (in seconds)

Ontology	Prep.+ Cons.	Classification				Realisation			
		No-C	Det-C	ET-C	LT-C	No-C	Det-C	ET-C	LT-C
OGSF	0.0	3.8	1.0	0.2	0.2	0.1	0.0	0.0	0.0
Wine	0.0	49.5	29.6	0.8	0.8	49.1	25.8	0.2	0.1
OBI	0.2	65.9	19.2	1.5	1.5	2.2	2.0	0.0	0.1
DOLCE	0.0	6.7	1.1	0.2	0.2	$\geq$ 300.0	5.2	0.1	0.1
USDA-5	4.1	0.8	1.0	1.0	0.8	$\geq$ 300.0	38.7	20.5	20.2
COSMO	0.6	$\geq$ 300.0	$\geq$ 300.0	42.2	11.2	n/a	n/a	11.2	19.9
DPC1	6.5	0.1	0.2	0.1	0.1	$\geq$ 300.0	53.3	19.4	20.5
Oly	4.0	0.2	0.1	0.0	0.1	$\geq$ 300.0	15.6	8.0	8.8
UOBM-1	6.7	240.6	4.8	1.3	1.1	$\geq$ 300.0	$\geq$ 300.0	$\geq$ 300.0	$\geq$ 300.0
CobCav92	18.5	58.0	11.5	11.5	11.5	101.9	1.7	0.8	0.8
LUBM-1	0.7	0.0	0.0	0.0	0.0	0.6	0.2	0.2	0.2

consistency checking as well as preprocessing are prerequisites of classification. Thus, realisation cannot be performed if the time limit is already reached for classification.

If no completion graph caching is activated (No-C), then the realisation and classification can often require a large amount of time since Konclude has to re-process the entire ABox for all instance and subsumption tests (if the ontology uses nominals). For several ontologies, such as Wine and DOLCE, the caching and re-use of the deterministic completion graph from the consistency check (Det-C) already leads to significant improvements. Nevertheless, with the two variants ET-C and LT-C of the presented completion graph caching technique, where ET-C uses an “early testing” and LT-C a “late testing” of the defined caching criteria, Konclude can further reduce the reasoning times. In particular, with both completion graph caching techniques, all evaluated ontologies can easily be classified and also the realisation can be realised efficiently for all but UOBM-1. For the realisation of UOBM-1, Konclude would require 951.2 s, which is primarily caused by axioms of the form PeopleWithManyHobbies $\equiv \geqslant 3 \text{like}.\top$, where it is required to merge the heavily connected nominal nodes that represent hobbies in order to test whether an individual is an instance of the class PeopleWithManyHobbies. Since the hobbies are not stated as different (and there are also no other restrictions that prevent the merging of these nodes), the individuals in UOBM-1 cannot be instances of PeopleWithManyHobbies, but, for the construction of each counter-model, many edges from individuals must be updated to the merged nominal nodes and, therefore, large parts of the cached completion graph become modified. Thus, the effectiveness of the completion graph caching is limited for UOBM-1. Table 3 also reveals that there is only for COMSO a remarkable difference between ET-C and LT-C, where this difference can be explained by the fact that there is often more interaction with the individuals from the ABox for instance tests than for satisfiability and subsumption tests and, therefore, ET-C can be better for realisation due to the reduced effort in combination with backtracking.

The effects of the different completion graph caching techniques can also be observed for the OWL DL Realisation dataset of the ORE 2014 competition,[8] which

[8] `http://www.easychair.org/smart-program/VSL2014/ORE-index.html`

contains several ontologies with non-deterministic language features and non-trivial ABoxes. By excluding $1,331$ s spent for preprocessing and consistency checking, the accumulated classification times over the contained 200 ontologies are $3,996$ s for the version No-C, $2,503$ s for Det-C, $1,801$ s for ET-C, and $1,606$ s for LT-C. Clearly, the dataset also contains many ontologies for which completion graph caching does not seem to have a significant impact, for example, if the ontologies can be processed deterministically or the ontologies are too difficult to even perform consistency checking (which is the case for 3 ontologies). Nevertheless, our completion graph caching improves the classification time with similar performances for LT-C and ET-C. By using the satisfiability caching extension for nominals, as presented in Section 4.1, the accumulated classification time can be further improved to 725 s. Similar results are also achieved for the realisation of these ontologies. By excluding the times for all prerequisites, the accumulated realisation times over all 200 ontologies are $1,740$ s for No-C, $1,498$ s for Det-C, $1,061$ s for ET-C, $1,256$ s for LT-C, and 923 s for the version where the satisfiability caching extension for nominals is additionally activated.

Unfortunately, a direct comparison with previously developed completion graph caching techniques or other approaches that reduce the ABox reasoning effort is not easily possible. For example, the original completion graph caching technique [22] that is integrated in the reasoning system Pellet [23] cannot be realised in Konclude since this would require that modifications for subsequent tests are added to the non-deterministic version of the cached completion graph from the initial consistency check, but such modifications would be lost after the backtracking from possible clashes. Also a comparison between Pellet and Konclude has only a limited validity since the reasoners have many different optimisations. However, for the ontologies depicted in Table 2, Pellet 2.3.1 reaches the time limit for the classification of all ontologies with nominals except Wine, for which Pellet requires 23.1 s. This can be seen as indication that the completion graph caching in Pellet does potentially not work as good as the presented technique that we realised in Konclude. In contrast, the recently proposed ABox absorption approach is implemented in the reasoner CARE,[9] but it does not provide a reasoning task that is also supported by Konclude.

6.1 Incremental Reasoning Experiments

To test the incremental reasoning based on the presented completion graph caching, we used those ontologies of Table 2 that have a large amount of ABox assertions and for which Konclude still has a clearly measurable reasoning time, i.e., COSMO, DPC1, USDA-5, Oly, CobCav92, UOBM-1, and (in order to demonstrate the effects for deterministic ontologies) LUBM-1. We simulated a changed ABox for these ontologies by randomly removing a certain amount of assertions from the ontology (denoted by $\mathcal{K}$) and by re-adding the removed assertions and removing new assertions (denoted by $\mathcal{K}^{\mp\Delta}$). For each ontology, we evaluated 10 random modifications that have 0.25, 0.5, 1, 2, 4, and 8 % of the size of the ontology's ABox. For simplicity, we only simulated modifications where the set of removed assertions and the set of added assertion have

[9] `http://code.google.com/p/care-engine/`

the same size. The obtained results for the presented incremental reasoning approach are shown in Table 4.

For consistency, the first two columns show the (incremental) consistency checking time (in seconds) for $\mathcal{K}$ and $\mathcal{K}^{\mp\Delta}$, respectively, and the last two columns show the percentage of the nodes in the completion graph for $\mathcal{K}$ that were directly and indirectly changed for the application of the modification Δ. Please note that the reasoner must be able to efficiently access a object property assertion $r(a, b)$ from both individuals, which means that also the data structure for b is updated if $r(a, b)$ has been added/removed and, therefore, the number of nodes that are perceived as directly changed is usually higher than the number of changed assertions. It can be observed that, especially for smaller modifications, the incremental consistency check often requires much less time than the initial consistency check. In particular, the individuals of USDA-5 are sparsely connected via object properties and, therefore, often only the modified individuals have to be re-built, which results in very low reasoning times for the incremental consistency check. For larger modifications, the incremental consistency checking time increases significantly for some ontologies, e.g., UOBM-1, and does almost catch up to the consistency checking time of the initial ontology. On the one hand, our incremental consistency checking approach has clearly some additional overhead for propagating the compatibility status and for expanding affected neighbour nodes step by step, but, on the other hand, our prototypical implementation has still a lot of room for improvements. For example, we currently choose the next neighbour node for required expansions randomly, but this can be improved by choosing a node to which new consequences are propagated or from which consequences are missing. This would significantly reduce the number of nodes that have to be re-built for the incremental consistency check. Moreover, we currently also re-build nodes for individuals for which only new assertions are added although it would be sufficient to simply extend the nodes of the previous deterministic completion graph by the new consequences.

For classification, the first two columns show analogously the (incremental) classification time for $\mathcal{K}$ and $\mathcal{K}^{\mp\Delta}$, respectively, the third column represents the percentage of the number of tested modifications for which re-classification was necessary, and the last column represents the percentage of the classes for which satisfiability and subsumption tests were re-calculated. At the moment, the change dependent nodes are tracked together for all satisfiability and subsumption tests and we additionally mark those classes for which nodes have been tracked. As a consequence, we have to recompute the satisfiability and subsumers of the marked classes if the tracked nodes have an overlapping with the (indirectly) changed nodes. This can, however, also be improved for smaller TBoxes by tracking the change dependent nodes for each class separately. Clearly, if the ontologies do not contain nominals, then re-classification is not required. But even some ontologies with nominals (e.g., UOBM-1), only a few classes have to be re-classified and, in several cases, re-classification is not required at all.

Also for realisation, the first two columns show the (incremental) realisation time for $\mathcal{K}$ and $\mathcal{K}^{\mp\Delta}$, respectively. The last two columns show the percentage of the number of re-computed individuals and possible instances that are potentially affected through the changes. In contrast to classification, we separately track for each individual the change dependent nodes. Therefore, we often have to re-compute only the possible instances for a part of the individuals, which often leads to a corresponding reduction in reasoning time.

Please note that, in practice, the ABox changes are often small. In particular, modifications that change 1 % of an ABox can, for several ontologies, already be considered as large since they can consist of several hundreds or even of thousands of axioms, e.g., a modification of 1 % of the size of the ABox of UOBM-1 already contains $2,605$ axioms. For many practical applications, however, only a few axioms are changed between the reasoning is invoked and our incremental reasoning approach (as well as many other incremental reasoning techniques) are designed for such cases.

Table 4. Incremental reasoning effort for different reasoning tasks on changed ABoxes (Reclas. stands for Reclassification, Recomp. for Recomputation, DCN stands for Directly Changed Nodes, ICN for Indirectly Changed Nodes, H for Hierarchy, C for Classes, I for Individuals, and PI for Possible Instances)

Ontology	$\frac{\lvert\Delta\rvert\cdot 100}{\lvert\mathcal{K}\rvert}$	Consistency				Classification				Realisation			
		Time [s]		Changes [%]		Time [s]		Reclas. [%]		Time [s]		Recomp. [%]	
		$\mathcal{K}$	$\mathcal{K}^{\mp\Delta}$	DCN	ICN	$\mathcal{K}$	$\mathcal{K}^{\mp\Delta}$	H	C	$\mathcal{K}$	$\mathcal{K}^{\mp\Delta}$	I	PI
USDA-5	0.25	3.1	0.0	0.0	0.0	0.9	–	–	–	18.2	0.0	1.0	1.0
USDA-5	0.50	3.3	0.0	0.0	0.0	0.8	–	–	–	18.1	0.0	0.5	0.5
USDA-5	1.00	3.1	0.0	0.0	0.0	0.9	–	–	–	18.2	0.0	1.0	1.0
USDA-5	2.00	3.2	0.0	0.0	0.0	0.8	–	–	–	14.9	0.0	2.0	2.0
USDA-5	4.00	3.2	0.1	0.0	0.0	0.8	–	–	–	17.0	0.0	3.9	3.9
USDA-5	8.00	3.2	0.1	0.1	0.1	0.9	–	–	–	15.6	0.1	7.9	7.9
COSMO	0.25	0.4	0.0	0.4	1.7	16.9	10.8	100.0	73.0	0.4	0.2	1.7	22.2
COSMO	0.50	0.4	0.0	0.8	1.5	26.1	16.1	100.0	73.3	0.5	0.3	1.6	22.0
COSMO	1.00	0.4	0.0	1.6	3.1	24.4	17.4	100.0	73.0	0.4	0.2	3.1	33.8
COSMO	2.00	0.4	0.1	3.1	5.5	25.9	18.0	100.0	73.0	0.5	0.3	5.6	44.9
COSMO	4.00	0.4	0.1	6.0	9.9	25.2	18.6	100.0	72.8	0.6	0.4	10.0	54.9
COSMO	8.00	0.4	0.2	11.4	16.6	20.6	17.1	100.0	73.0	0.5	0.6	17.1	88.5
DPC1	0.25	4.9	0.2	0.1	0.5	0.1	–	–	–	27.0	4.6	3.5	5.7
DPC1	0.50	5.1	0.4	0.3	0.8	0.1	–	–	–	28.7	7.2	5.2	8.3
DPC1	1.00	5.0	0.6	0.5	1.5	0.1	–	–	–	29.1	14.0	10.0	15.6
DPC1	2.00	4.9	1.1	1.0	2.6	0.1	–	–	–	27.7	19.9	16.9	25.4
DPC1	4.00	5.0	2.0	1.9	4.4	0.1	–	–	–	29.3	26.7	26.5	38.0
DPC1	8.00	5.0	3.1	3.4	7.3	0.1	–	–	–	34.2	36.7	51.8	54.6
Oly	0.25	3.1	0.1	0.1	0.5	0.1	–	–	–	8.3	1.6	3.6	6.0
Oly	0.50	3.2	0.2	0.3	0.8	0.1	–	–	–	10.9	2.7	5.2	8.4
Oly	1.00	3.2	0.4	0.5	1.4	0.1	–	–	–	10.4	4.5	9.3	14.9
Oly	2.00	3.1	0.7	1.0	2.8	0.1	–	–	–	10.1	7.4	17.4	26.8
Oly	4.00	3.2	1.3	1.9	4.8	0.1	–	–	–	12.2	11.4	28.0	40.2
Oly	8.00	3.1	1.9	3.6	7.7	0.1	–	–	–	11.6	14.2	43.5	56.5
UOBM-1	0.25	3.5	1.6	1.9	6.7	1.2	0.4	10.0	2.6	≥ 300.0		*n/a*	*n/a*
UOBM-1	0.50	3.6	1.9	2.7	8.4	1.4	0.6	20.0	3.6	≥ 300.0		*n/a*	*n/a*
UOBM-1	1.00	3.6	2.3	3.3	10.0	1.3	0.8	30.0	5.9	≥ 300.0		*n/a*	*n/a*
UOBM-1	2.00	3.7	2.9	5.8	14.0	1.4	1.1	40.0	7.9	≥ 300.0		*n/a*	*n/a*
UOBM-1	4.00	3.7	3.5	9.6	18.2	1.4	1.7	60.0	11.3	≥ 300.0		*n/a*	*n/a*
UOBM-1	8.00	3.7	3.8	14.1	21.6	1.5	2.0	70.0	13.4	≥ 300.0		*n/a*	*n/a*
CobCav92	0.25	10.8	0.8	1.3	4.9	0.0	0.0	0.0	0.0	1.6	1.2	4.2	0.0
CobCav92	0.50	11.3	1.4	2.5	8.6	0.0	0.0	10.0	0.0	1.5	1.2	7.9	0.0
CobCav92	1.00	11.1	2.7	4.6	15.4	0.0	0.0	10.0	0.0	1.7	1.3	14.8	0.0
CobCav92	2.00	10.5	4.4	8.1	25.8	0.0	0.0	10.0	0.0	1.5	1.4	25.9	0.0
CobCav92	4.00	11.6	7.3	12.9	38.6	0.0	0.3	20.0	0.1	1.7	1.6	40.7	0.0
CobCav92	8.00	11.6	9.7	18.1	50.4	0.0	0.8	50.0	0.1	1.7	1.7	54.9	0.0
LUBM-1	0.25	1.4	0.1	0.7	7.0	0.0	–	–	–	0.1	0.1	4.0	0.0
LUBM-1	0.50	1.4	0.2	1.4	1.5	0.0	–	–	–	0.1	0.1	6.9	0.0
LUBM-1	1.00	1.4	0.3	2.6	4.9	0.0	–	–	–	0.1	0.1	12.5	0.0
LUBM-1	2.00	1.3	0.5	4.9	8.1	0.0	–	–	–	0.1	0.1	21.0	0.0
LUBM-1	4.00	1.4	0.9	8.8	13.1	0.0	–	–	–	0.2	0.1	34.6	0.0
LUBM-1	8.00	1.4	1.2	14.5	19.7	0.0	–	–	–	0.2	0.2	53.7	0.0

6.2 Representative Caching Experiments

We integrated a first prototypical version of the presented representative caching in our reasoning system Konclude, which is, however, not yet compatible with all other integrated features and optimisations. As of now, the integrated representative caching is primarily used for "simple individuals" that do not have too much interaction with other individuals in the ABox. In cases where representative caching could easily cause performance deficits (e.g., through the intensive use of nominals), Konclude caches the relevant parts of such completion graphs by using the ordinary technique. Moreover, data property assertions are, at the moment, internally transformed into class assertions and, as a consequence, nodes for individuals with data property assertions can currently not be cached in a representative way.

However, first experiments are very encouraging. For example, consistency checking of LUBM-1 without data property assertions requires only 16 MB by using the representative caching, whereas a fully expanded completion graph constructed by the tableau algorithm requires 481 MB in Konclude. As comparison, the internal representation of LUBM-1 requires 44 MB and 46 MB are additionally used for the parsed OWL objects, which are kept in memory to facilitate the handling of added/removed axioms and to efficiently answer queries about told axioms since the preprocessing (e.g., absorption, lexical normalisation) often changes the internal representations significantly for ontologies that use more expressive language features. The doubled representation could, however, easily be avoided for ABox assertions since they have a very simple structure and, as a consequence, the overall memory consumption for ontologies with large ABoxes could be further improved. Since the representative caching is only used for the simple individuals, the reasoning performance is hardly affected.

Similar improvements can also be achieved for ontologies with more expressive language features. In particular, Homo_sapiens[10] is a very large $\mathcal{SROIQ}$ ontology from the Oxford ontology library with $244,232$ classes, 255 object properties, and $289,236$ individuals for which Konclude requires $10,211$ MB in total to check the consistency by using the representative caching, whereas $19,721$ MB are required by Konclude for consistency checking with a fully expanded completion graph. With the representative caching, the majority of the memory consumption is again caused by the internal representation and the data from the preprocessing ($9,875$ MB). Nevertheless, the classification of Homo_sapiens still requires approximately 707.0 s for Konclude. This can potentially be improved by further extending the presented satisfiability caching with nominal support. For example, one could also allow the caching of node labels even if the dependent nominal nodes in the cached completion graph are modified by additionally storing in the cache which entries are compatible with each other, i.e., which cached labels have occurred in the same completion graph. Hence, different cache entries that would influence nominal nodes can safely be re-used if the dependent nominal nodes do not overlap or the cache entries are compatible with each other.

[10] `http://www.cs.ox.ac.uk/isg/ontologies/lib/OMEO/Homo_sapiens.owl/`

7 Conclusions and Future Work

We have presented a new technique to improve ABox reasoning with expressive Description Logics. This is achieved by caching the completion graph of the initial ABox consistency test and re-using this data in subsequent tests. In particular, we only have to re-process parts of the ABox until an expansion as for the initial consistency check can be guaranteed for the remaining parts of the ABox.

This caching technique was essential for the good realisation performance of our reasoning system Konclude at the ORE 2014 competition, where Konclude was able to outperform other reasoners in the OWL EL as well as in the OWL DL discipline. As also confirmed by the presented evaluation, the technique reduces the ABox reasoning effort for all reasoning tasks for which consequences of the ABox have potentially to be considered. Moreover, the technique is well-suited for integration into other tableau-based reasoning systems since it does not require significant adaptations to the reasoner and also does not produce a significant overhead. For example, if a knowledge base does not contain any individuals, then reasoning is not affected at all.

We also presented and evaluated extensions and applications of the caching technique, which allow for supporting nominals for the satisfiability caching of node labels, for reducing reasoning effort for incrementally changed ABoxes, and for handling very large ABoxes by storing partially processed parts of the completion graph in a representative way.

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. 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)
2. 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)
3. Cuenca Grau, B., Halaschek-Wiener, C., Kazakov, Y., Suntisrivaraporn, B.: Incremental classification of description logics ontologies. J. of Automated Reasoning 44(4), 337–369 (2010)
4. Dolby, J., Fokoue, A., Kalyanpur, A., Schonberg, E., Srinivas, K.: Scalable highly expressive reasoner (SHER). J. of of Web Semantics 7(4), 357–361 (2009)
5. Donini, F.M., Massacci, F.: EXPTIME tableaux for $\mathcal{ALC}$. J. of Artificial Intelligence 124(1), 87–138 (2000)
6. Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: Hermit: An OWL 2 reasoner. J. of Automated Reasoning 53(3), 1–25 (2014)
7. Glimm, B., Kazakov, Y., Liebig, T., Tran, T.K., Vialard, V.: Abstraction refinement for ontology materialization. In: Proc. 13th Int. Semantic Web Conf. (ISWC'14). LNCS, vol. 8797, pp. 180–195. Springer (2014)

8. Guo, Y., Pan, Z., Heflin, J.: Lubm: A benchmark for OWL knowledge base systems. J. of Web Semantics 3(2-3), 158–182 (2005)
9. Haarslev, V., Möller, R.: On the scalability of description logic instance retrieval. J. of Automated Reasoning 41(2), 99–142 (2008)
10. 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)
11. Halaschek-Wiener, C., Hendler, J.: Toward expressive syndication on the web. In: Proc. 16th Int. Conf. on World Wide Web (WWW'07). ACM (2007)
12. Halaschek-Wiener, C., Parsia, B., Sirin, E.: Description logic reasoning with syntactic updates. In: Proc. 4th Confederated Int. Conf. On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE(OTM'06), LNCS, vol. 4275, pp. 722–737. Springer (2006)
13. 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)
14. 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)
15. 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)
16. 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)
17. 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)
18. 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)
19. Kazakov, Y., Klinov, P.: Incremental reasoning in OWL EL without bookkeeping. In: Proc. 12th Int. Semantic Web Conf. (ISWC'13). LNCS, vol. 8218, pp. 232–247. Springer (2013)
20. Kollia, I., Glimm, B.: Optimizing SPARQL query answering over OWL ontologies. J. of Artificial Intelligence Research 48, 253–303 (2013)
21. Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y., Liu, S.: Towards a complete OWL ontology benchmark. In: Proc. 3rd European Semantic Web Conf. (ESWC'06). LNCS, vol. 4011, pp. 125–139. Springer (2006)
22. Sirin, E., Cuenca Grau, B., Parsia, B.: From wine to water: Optimizing description logic reasoning for nominals. In: Proc. 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'06). pp. 90–99. AAAI Press (2006)
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. 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)
25. Steigmiller, A., Glimm, B., Liebig, T.: Optimised absorption for expressive description logics. In: Proc. 27th Int. Workshop on Description Logics (DL'14). vol. 1193. CEUR (2014)
26. Steigmiller, A., Liebig, T., Glimm, B.: Konclude: system description. J. of Web Semantics 27(1) (2014)
27. Tsarkov, D., Horrocks, I.: Efficient reasoning with range and domain constraints. In: Proc. 17th Int. Workshop on Description Logics (DL'04). vol. 104. CEUR (2004)
28. Tsarkov, D., Horrocks, I., Patel-Schneider, P.F.: Optimizing terminological reasoning for expressive description logics. J. of Automated Reasoning 39, 277–316 (2007)

29. 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/`
30. Wandelt, S., Möller, R.: Towards ABox modularization of semi-expressive description logics. Applied Ontology 7(2), 133–167 (2012)
31. Wu, J., Hudek, A.K., Toman, D., Weddell, G.E.: Absorption for ABoxes. J. of Automated Reasoning 53(3), 215–243 (2014)
32. Wu, J., Lecue, F.: Towards consistency checking over evolving ontologies. In: Proc. 23rd ACM Int. Conf. on Information and Knowledge Management (CIKM'14). pp. 909–918. ACM (2014)