Nominal Schema Absorption
Technical Report

Andreas Steigmiller, Birte Glimm, Thorsten Liebig

Ulmer Informatik-Berichte
Nr. 2013-06
Juni 2013
Nominal Schema Absorption
Technical Report

Andreas Steigmiller¹, Birte Glimm¹, and Thorsten Liebig²

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

Abstract. Nominal schemas have recently been introduced as a new approach for the integration of DL-safe rules into the Description Logic framework. The efficient processing of knowledge bases with nominal schemas remains, however, challenging. We address this by extending the well-known optimisation of absorption as well as the standard tableau calculus to directly handle the (absorbed) nominal schema axioms. We implement the resulting extension of standard tableau calculi in the novel reasoning system Konclude and present further optimisations. In our empirical evaluation, we show the effect of these optimisations and we find that the proposed nominal schema handling performs well even when compared to (hyper)tableau systems with dedicated rule support.

1 Introduction

We address the problem of an efficient handling of so-called nominal schema axioms in tableau calculi for Description Logics (DLs). Nominal schemas have been introduced recently [14] as a feature for expressing arbitrary DL-safe rules (as specified in the W3C standards SWRL [7] or RIF [12]) natively in DLs and, consequently, in OWL ontologies [17]. Hence, DLs with nominal schemas provide a unified basis for OWL and rules. Although some attempts (see, e.g., [13]) have been made to improve the performance of tableau calculi when extended with nominal schemas, handling of nominal schemas remains challenging. We tackle this problem by extending the well-know tableau optimisation of absorption [10]. The resulting calculus extends a standard tableau calculus by additional rules to deal with the absorbed nominal schema axioms and shows a considerable performance improvement over existing techniques.

Nominal schemas extend the nominal constructor that is present in many DLs and which allows for specifying a concept as a singleton set with a named individual as member, e.g., the interpretation of the concept \{a\} consists of the element that represents the named individual a. Nominal schemas introduce a new concept constructor \{x\}, where x is a variable that can only be bound to a named individual from the ABox of the knowledge base. This restriction ensures decidability and is common for nominal schemas as well as for SWRL rules.

We use the same running example (or parts thereof) as Krisnadhi and Hitzler [13], which describes a conflicting review assignment between a person and a paper if the individual has to review a paper x that has an author (y) with whom that individual has
a joint publication in the same venue (z):

\[ \exists \text{hasReviewAssignment}.(\{x\} \sqcap \exists \text{hasAuthor}.\{y\} \sqcap \exists \text{atVenue}.\{z\}) \]
\[ \sqcap \exists \text{hasSubmittedPaper}.(\exists \text{hasAuthor}.\{y\} \sqcap \exists \text{atVenue}.\{z\}) \]
\[ \sqsubseteq \exists \text{hasConflictingAssignedPaper}.\{x\}. \]

For brevity, we shorten hasReviewAssignment to r, hasAuthor to a, atVenue to v, hasSubmittedPaper to s, and hasConflictingAssignedPaper to c in the remainder. Obviously, this axiom can neither be directly expressed in a DL knowledge base nor as ordinary DL-safe rule (e.g., if we were to express the complex concepts as role atoms, we would have to introduce a variable for the submitted paper, which then would only bind to known ABox individuals). However, such nominal schema axioms can easily be eliminated by replacing them with all corresponding grounded axioms, i.e., the axioms that are obtained by replacing each nominal schema by a nominal, in all possible combinations, where all nominal schemas with the same variable are replaced by the same nominal. Thus, a knowledge base for a DL with nominal schema constructs, which is denoted by an additional 'V' in the DL nomenclature, can be reduced, with this upfront grounding approach, to a knowledge base without nominal schema axioms. The upfront grounding is, however, very inefficient. For example, a nominal schema axiom with 3 variables can be grounded for a knowledge base with 100 ABox individuals in 100\(^3\) different ways, which is prohibitive even for small examples. One way to restrict the effort of reasoning with nominal schemas is to restrict the expressiveness of the nominal schema axioms, whereby it is possible to achieve that the grounding adds only linearly or polynomially many new axioms [14].

For efficient reasoning in OWL ontologies extended with nominal schemas, i.e., SROIQV knowledge bases, it is more promising to adapt the established tableau algorithms, which are dominantly used for sound and complete reasoning systems. One such approach extends a tableau algorithm such that grounding is delayed until it is required [13]. The standard rules are blocked until the new grounding rules ensure that a concept with nominal schemas can be processed safely, e.g., the concept \(\exists r.\{x\} \sqcap C\) has to be grounded before the \(\exists\)-rule can be applied. However, this requires significant changes to the tableau algorithm and, thus, existing optimisations, which are crucial for a reasonable performance on real-world ontologies, have to be adapted as well. Furthermore, it is not clear in which way concepts have to be grounded to achieve a well-performing implementation and some concepts even cannot be grounded efficiently, e.g., disjunctions that have the same nominal schema variable in several disjuncts have to be grounded before the disjunction can be processed.

In this paper, we present a new approach that works more from the opposite direction by collecting possible bindings for the nominal schema variables during the application of rules and, then, these bindings are used to complete the processing of the nominal schema axioms. To implement this idea, we extend the absorption, which is a widely used preprocessing step (Section 2.3 and 3), to handle nominal schemas (Section 4.1), and we adapt or add new rules to the tableau calculus, which create and propagate bindings of variables through the completion graph constructed by the tableau algorithm (Section 4.2). These bindings are then used to ground the remaining, non-absorbable part of the nominal schema axioms. Our rules can be completely separated
from other standard rules and, thus, can be integrated well into existing implementations without any adaptation of other optimisations. We have implemented our nominal schema absorption technique into the novel SROIQ reasoning system Konclude and the empirical evaluation shows that our approach works well, even if we convert ordinary DL-safe rules to nominal schema axioms and compare our approach to other DL reasoners with dedicated rule support.

The paper is organised as follows: Section 2 summarises some preliminaries about model construction calculi and absorption. In Section 4 we present our new nominal schema absorption technique, which is then further optimised in Section 5 and Section 6. The empirical evaluation is shown in Section 7 and we conclude in Section 8.

2 Preliminaries

In this section, we first give a brief introduction into Description Logics extended with nominal schemas. For ease of presentation, we introduce only the DL ALCOIQ with its extension to nominal schemas here instead of SROIQ, which underpins OWL 2, but our technique to handle nominal schemas can easily be used with SROIQ too, since we simply add a number of additional rules to an existing tableau calculus.

2.1 The Description Logic ALCOIQV

We first define the syntax and semantics of roles, and then go on to ALCOIQV-concepts, individuals, and ontologies/knowledge bases.

Definition 1 (Syntax of ALCOIQV). Let NC, NR, NI, and NV be countable, infinite, and pairwise disjoint sets of concept names, role names, individual names, and variable names, respectively. We call S = (NC, NR, NI, NV) a signature. The set rol(S) of ALCOIQV-roles over S (or roles for short) is NR ∪ {r− | r ∈ NR}, where roles of the form r− are called inverse roles.

The set of ALCOIQV-concepts (or concepts for short) over S is the smallest set built inductively over symbols from S using the following grammar, where a ∈ NI, x ∈ NV, n ∈ N0, A ∈ NC, and r ∈ rol(S):

C ::= ⊤ | ⊥ | {a} | {x} | A | ¬C | C1 ∩ C2 | C1 ⊔ C2 | ∀r.C | ∃r.C | ⩽n r.C | ⩾n r.C.

Definition 2 (Semantics of ALCOIQV-concepts). An interpretation I = (ΔI, ⊨I) consists of a non-empty set ΔI, the domain of I, and a function ⊨I, which maps every concept name A ∈ NC to a subset AI ⊆ ΔI, every role name r ∈ NR to a binary relation rI ⊆ ΔI × ΔI, and every individual name a ∈ NI to an element aI ∈ ΔI. For each role name r ∈ NR, the interpretation of its inverse role (r−)I consists of all pairs ⟨δ, δ′⟩ ∈ ΔI × ΔI for which ⟨δ′, δ⟩ ∈ rI. A variable assignment for I is a function µ : NV → ΔI such that, for each x ∈ NV, µ(x) = aI for some a ∈ NI.
For any interpretation \( I \) and assignment \( \mu \), the semantics of \( \mathcal{ALCIOQ}V \)-concepts over a signature \( S \) is defined by the function \( \cdot \mu \) as follows:

\[
\begin{align*}
\top \mu &= \Delta^f \\
\bot \mu &= \emptyset \\
A \mu &= \Delta^f \\
r \mu &= r^f \\
(C \sqcap D) \mu &= C^f \mu \sqcap D^f \mu \\
(C \sqcup D) \mu &= C^f \mu \sqcup D^f \mu \\
(\forall r.C) \mu &= \{ \delta \in \Delta^f | \text{if } (\delta, \delta') \in r^f \text{ then } \delta' \in C^f \mu \} \\
(\exists r.C) \mu &= \{ \delta \in \Delta^f | \exists \delta' \in C^f \mu \text{ with } (\delta, \delta') \in r^f \} \\
(\leq n r.C) \mu &= \{ \delta \in \Delta^f | \exists \delta' \in C^f \mu \text{ such that } (\delta, \delta') \in r^f \} \\
(\geq n r.C) \mu &= \{ \delta \in \Delta^f | \forall \delta' \in C^f \mu \text{ such that } (\delta, \delta') \in r^f \}
\end{align*}
\]

where \( \#M \) denotes the cardinality of the set \( M \).

**Definition 3 (Syntax and Semantics of Axioms and Ontologies).** For \( C, D \) concepts, a general concept inclusion (GCI) is an expression \( C \subseteq D \). We introduce \( C \equiv D \) as an abbreviation for \( C \subseteq D \) and \( D \subseteq C \). A finite set of GCIs is called a TBox. 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 pair \( (\mathcal{T}, \mathcal{A}) \) with \( \mathcal{T} \) a TBox and \( \mathcal{A} \) an ABox.

Let \( I = (\Delta^f, \cdot^f) \) be an interpretation and \( \mu \) an assignment, then \( I \) and \( \mu \) satisfy an axiom or assertion \( \alpha \), written \( I, \mu \models \alpha \) if (i) \( \alpha \) is a GCI \( C \subseteq D \) and \( C^f \mu \subseteq D^f \mu \), (ii) \( \alpha \) is an assertion \( C(a) \) and \( a^f \in C^f \mu \) or (iii) \( \alpha \) is an assertion \( r(a, b) \) and \( (a^f, b^f) \in r^f \).
The interpretation \( I \) satisfies \( \alpha \) if \( I, \mu \models \alpha \) for every assignment \( \mu \). \( I \) satisfies a TBox \( \mathcal{T} \) (ABox \( \mathcal{A} \)) if it satisfies each GCI in \( \mathcal{T} \) (each assertion in \( \mathcal{A} \)). We say that \( \mathcal{K} \) satisfies \( \mathcal{T} \) if \( \mathcal{I} \) satisfies \( \mathcal{T} \) and \( \mathcal{A} \). In this case, we say that \( I \) is a model of \( \mathcal{K} \) and write \( I \models \mathcal{K} \). We say that \( \mathcal{K} \) is consistent if \( \mathcal{K} \) has a model.

Note, if the knowledge base does not contain any nominal schemas, then we do not have to consider the variable assignments for the satisfiability of axioms, TBoxes, ABoxes and knowledge bases.

### 2.2 Tableau Calculus

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 4 (Completion Graph).** A completion graph for \( \mathcal{K} \) is a directed graph \( G = (V, E, \mathcal{L}) \). Each node \( v \in V \) (edge \( (v, v') \in E \)) is labelled with a set of concepts (roles) occurring in \( \mathcal{K} \). The symmetric binary relation \( \approx \) is used to keep track of inequalities between nodes in \( V \).

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

**Definition 5 (Successor, Predecessor, Neighbour).** If \( (v_1, v_2) \in E \), 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 \(r\)-successor of a node \(v_1\) if \(r \in \mathcal{L}(v_1, v_2)\); \(v_2\) is called an \(r\)-predecessor of \(v_1\) if \(v_1\) is an \(r\)-successor of \(v_2\). A node \(v_2\) is called a neighbour (\(r\)-neighbour) of a node \(v_1\) if \(v_2\) is a successor (\(r\)-successor) of \(v_1\) or if \(v_1\) is a successor (\(r\)-successor) of \(v_2\).

The completion graph is initialised for the tableau algorithm by creating one node for each ABox individual/nominal in the input knowledge base (w.l.o.g. we assume that the ABox is non-empty, should this not be the case, we can always add an assertion \(\top\) for a fresh individual \(a\)) and by adding the concept and role facts for the ABox assertions of \(\mathcal{K}\). If \(v_1, \ldots, v_L\) are the nodes for the ABox individuals \(a_1, \ldots, a_L\) of \(\mathcal{K}\), then the initial completion graph \(G = (V, E, L, \emptyset)\) has to contain (i) for each ABox assertion of the form \(C(a_i)\) the concept fact \(C(v_i)\), i.e., \(C \in L(v_i)\), (ii) for each ABox assertion of the form \(r(a_i, a_j)\) the role fact \(r(v_i, v_j)\), i.e., \((v_i, v_j) \in E \text{ and } r \in \mathcal{L}((v_i, v_j))\). Furthermore, we add for each ABox individual \(a_i\) the nominal \(\{a_i\}\) and the concept \(\top\) to the label of \(v_i\), i.e., \(\{a_i\}, \top \subseteq L(v_i)\).

Additionally, we assume all concepts to be 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 duality between existential and universal restrictions, and between atmost and atleast number restrictions [9]. For \(C\) a concept possibly not in NNF, let \(\text{nff}(C)\) be the equivalent concept to \(C\) in NNF.

The tableau algorithm works by decomposing concepts in the completion graph with a set of expansion rules (see Table 1). Note, Table 1 shows only the expansion rules for \(\mathcal{ALC}\), i.e., to support the concept constructors of more expressive Description Logics, we have to add the corresponding expansion rules, e.g., for \(\mathcal{ALCOIQ}\) also the \(\sqcap, \sqcup, \preceq, \succeq\) and \(\text{NN}\)-rules are necessary. However, these rules are not affected by our extension for handling absorbed nominal schema axioms and, thus, we omit their presentation here.

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

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\sqsubseteq)-rule</td>
<td>if (A \in L(v), A \sqsubseteq C \in \mathcal{K}, v) not indirectly blocked, and ({C} \not\subseteq L(v)) then (L(v) \rightarrow L(v) \cup {C})</td>
</tr>
<tr>
<td>(\sqcap)-rule</td>
<td>if (C_1 \sqcap C_2 \in L(v), v) not indirectly blocked, and ({C_1, C_2} \not\subseteq L(v)) then (L(v) \rightarrow L(v) \cup {C_1, C_2})</td>
</tr>
<tr>
<td>(\sqcup)-rule</td>
<td>if (C_1 \sqcup C_2 \in L(v), v) not indirectly blocked, and ({C_1, C_2} \cap L(v) = \emptyset) then (L(v) \rightarrow L(v) \cup {[H] \text{ for some } H \in {C_1, C_2}})</td>
</tr>
<tr>
<td>(\exists)-rule</td>
<td>if (\exists r.C \in L(v), v) not blocked, and (\bar{n}) has no (r)-neighbour (v') with (C \in L(v')) then create new node (v') and edge ((v, v')) with (L(v') := {\top, C}) and (L((v, v')) := {r})</td>
</tr>
<tr>
<td>(\forall)-rule</td>
<td>if (\forall r.C \in L(v), v) not indirectly blocked, and there is an (r)-neighbour (v') of (v) with ({C} \not\subseteq L(v')) then (L(v') \rightarrow L(v') \cup {C})</td>
</tr>
</tbody>
</table>

Table 1. Basic expansion rules for \(\mathcal{ALC}\) TBoxes
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.

Unrestricted application of the $\exists$-rule and $\geq$-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 \[8\] 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 6 (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 a node $w$ with predecessor $w'$ such that

1. $v, v', w, w'$ are all blockable,
2. $w, w'$ are not blocked,
3. $L(v) = L(w)$ and $L(v') = L(w')$,
4. $L((v', v)) = L((w', w))$.

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

In order to guarantee that each node of the completion graph indeed satisfies all axioms of the TBox, one can “internalise” the TBox into a concept that is added to each node label. For example, if the TBox contains the axioms

$$A_1 \sqsubseteq \exists r.(B_1 \sqcap B_2) \quad (1)$$

$$\exists r.(B_1 \sqcup B_2) \sqsubseteq \exists s.A_2 \quad (2)$$

the internalised concept $C_I$ contains one conjunct for each axiom that is a disjunction with the negated left-hand side of the axiom and the right-hand side:

$$C_I = \text{nnf}(((\neg A_1 \sqcup \exists r.(B_1 \sqcap B_2)) \sqcap (\neg(\exists r.(B_1 \sqcup B_2)) \sqcup \exists s.A_2))).$$

A tableau algorithm based on the expansion rules of Table 1 indirectly adds the internalised concept with an auxiliary axiom $\top \sqsubseteq C_I$ by the $\sqsubseteq_1$-rule, since the concept $\top$ is added to every node of a completion graph. Clearly, internalisation introduces a large number of disjunctions in each node label, which possibly require several non-deterministic choices and backtracking if the choice resulted in a clash.

It would be more efficient to integrate a rule into the tableau calculus that checks for each GCI $C \sqsubseteq D$ whether $C$ is satisfied for a node and if this is the case, then $D$ is added to the label of the node. For Axiom (1), for example, it is easy to check whether the atomic concept $A_1$ is satisfied at a node and if this is the case, then $\exists r.(B_1 \sqcap B_2)$ has to be added to the node label. This lazy unfolding for atomic concepts is already realised with the $\sqsubseteq_1$-rule of Table 1, whereby we do not have to internalise axioms of the form $A \sqsubseteq C$.

Checking whether a complex left-hand side of an axiom is satisfied can, however, be non-trivial. For example, it can often not be verified syntactically, whether a node
satisfies $\exists r (B_1 \sqcup B_2)$, which would be required for Axiom (2). For example, any instance of $A_1$ has an $r$-successor that satisfies $B_1 \cap B_2$ and, therefore, this $A_1$ instance (semantically) also satisfies $\exists r (B_1 \sqcup B_2)$. In order to, nevertheless, avoid the handling of $\top \sqsubseteq \text{nnf} (\neg C \sqcup D)$ one uses elaborate transformations in practise. Such transformations are performed in a preprocessing step called absorption, which we describe in more detail in Section 2.3 and 3.

2.3 Absorption

The absorption algorithm extracts those conditions of a disjunction for which it can be ensured that if one of these conditions is not satisfied for a node in a completion graph, then at least one alternative of the disjunction is trivially satisfiable. The extracted conditions are then used for expressing the disjunction in such a way that non-determinism can be avoided as much as possible in the tableau algorithm. For example, one would like to avoid treating Axiom (2) as $\top \sqsubseteq \forall r (\neg B_1 \cap \neg B_2) \sqcup \exists s A_2$ as motivated in the previous section. Any node that does not have an $r$-neighbour trivially satisfies $\forall r (\neg B_1 \cap \neg B_2)$ and, hence, the overall disjunction. Thus, we could only add the disjunction to nodes that have at least one $r$-successor. We can, however, go even further by first identifying nodes that satisfy $B_1$ or $B_2$. If we find such a node, we can make sure that its $r^\prec$-neighbour has to satisfy $\exists s A_2$. This is captured by the following axioms:

$$B_1 \sqsubseteq T \quad B_2 \sqsubseteq T \quad T \sqsubseteq \forall r^\prec (\exists s A_2),$$

where $T$ is a fresh atomic concept. Here, $B_1$ and $B_2$ have been absorbed (i.e., moved to the left-hand side of the axiom) and the concept $T$ is used to enforce the semantics of the original axiom. We call $\forall r (\neg B_1 \cap \neg B_2)$ completely absorbable since it no longer contributes a disjunct. The goal of the absorption preprocessing step is, therefore, the extraction of such easy to verify conditions that allow for expressing a GCI by possibly several inclusion axioms that ideally do not require a disjunction, e.g., as in the case of Axiom (2) above.

For the absorption of more complex concepts it is often necessary to join several conditions, say $A_1$ to $A_n$. A possibility to do this in an efficient way is binary absorption [11], where two concepts $A_1$ and $A_2$ imply a new concept $T_1$ by the axiom $(A_1 \cap A_2) \sqsubseteq T_1$. We can then combine $T_1$ with the next condition $A_3$ and so on, until $(T_{n-2} \cap A_n) \sqsubseteq T_{n-1}$, where $T_{n-1}$ can then be used for further absorption or to initiate the addition of the remaining and non-absorbed part of the disjunction. By joining the conditions binary, it is possible to reuse more of these joins for several axioms if the axioms have some common conditions. Note, a binary absorption axiom $(A_1 \cap A_2) \sqsubseteq C$ is usually handled by a separate $\sqsubseteq \sqcup$-rule, which adds the concept $C$ to a label only if $A_1$ and $A_2$ are already present.

3 Absorption Algorithm

Since our handling of nominal schemas is based on absorption methods, we next present an improved variant of a recursive binary absorption algorithm, which we then extend
In the following, the absorption of axioms of a (global) TBox is considered to be global for the ease of presentation:

- \( \text{isCA}(C) \) (isPA\((C)\)), shown in Algorithm 1, returns whether the concept \( C \) is completely (partially) absorbable. Note, if a concept \( C \) is completely absorbable, then it is also partially absorbable, and moreover, if a concept \( C \) is not partially absorbable, then it is also not completely absorbable. We have tagged the lines 3, 7 and 9 with \( n \).

```
Algorithm 1 isCA(C) and isPA(C)

Output: Returns whether the concept C is completely absorbable
1: procedure isCA(C)
2: if C = C1 \cup C2 then
3: return isCA(C1) \land isCA(C2) 
4: else if C = C1 \cap C2 then
5: return isCA(C1) \land isCA(C2)
6: else if C = \exists r.C' then
7: return isCA(C') 
8: else if C = \leq n r.C' then
9: return false 
10: else if C = \neg[A] then
11: return true 
12: else if C = \neg[A] then
13: if A is not acyclic then
14: return false 
15: end if 
16: for all A \equiv C' \in \mathcal{T} do
17: if \neg isCA(nn\nf(-C')) then
18: return false 
19: end if 
20: end for
21: return true
22: end if 
23: return false
24: end procedure
```

```
Algorithm 2 collectDisjuncts($C, absorbable$)

Output: Returns the absorbable/not absorbable disjuncts of the concept $C$

1: $S \leftarrow \{C\}$
2: while $(C_1 \cup C_2) \in S$ do
3:     $S \leftarrow (S \setminus (C_1 \cup C_2)) \cup \{C_1, C_2\}$
4: end while
5: if $absorbable$ then
6:     return $\{C \in S \mid \text{isPA}(C)\}$
7: else
8:     return $\{C \in S \mid \text{isCA}(C)\}$
9: end if

Algorithm 3 absorbJoined($S$)

Output: Returns the atomic concept that is implied by the join of the absorptions of $S$

1: $S' \leftarrow \emptyset$
2: for all $C \in S$ do
3:     $A' \leftarrow \text{absorbConcept}(C)$
4:     $S' \leftarrow S' \cup \{A'\}$
5: end for
6: while $A_1 \in S'$ and $A_2 \in S'$ and $A_1 \neq A_2$ do
7:     $T \leftarrow$ fresh atomic concept
8:     $T' \leftarrow T' \cup \{(A_1 \cap A_2) \subseteq T\}$
9:     $S' \leftarrow (S' \cup \{T\}) \setminus \{A_1, A_2\}$
10: end while
11: if $S' = \emptyset$ then return $\top$
12: else return the element $A' \in S'$
13: end if

A comment symbol to highlight where $\text{isPA}$ might allow additional absorption in comparison to $\text{isCA}$. In order to avoid an infinite recursion, we require that the concepts are acyclic for the absorption, where acyclicity is defined as follows: $A_1$ directly uses $A_2$ w.r.t. a TBox $T$ if $A_1 \equiv C \in T$ or $A_1 \subseteq C \in T$ and $A_2$ occurs in $C$; uses is the transitive closure of “directly uses”. Then, a concept $D$ is acyclic w.r.t. a TBox $T$ if it contains no concept $A$ that uses itself. We use the acyclicity restriction to keep the absorption algorithm simple, however, this restriction is not relevant in practice, because a cyclic concept $A$ with the definition $A \subseteq C$ can simply be made acyclic by representing $A \subseteq C$ as $\top \subseteq \neg A \cup C$.

• collectDisjuncts($C, absorbable$), shown in Algorithm 2, returns a set of (partially or completely) absorbable disjuncts for a concept $C$ if $absorbable = \text{true}$ and a set of not completely absorbable disjuncts otherwise. If $C$ is not a disjunction, then $\{C\}$ itself is returned, in case it conforms to the specified $\text{absorbable}$ condition.

The absorption itself is invoked by the absorbTBox procedure (see Algorithm 5). Each axiom of $T$ is processed and the resulting axioms are added to $T'$. For a possibly absorbable axiom the set of all absorbable disjuncts is extracted with collectDisjuncts from the corresponding disjunction and then absorbJoined is called for generating the absorption. Please note that if all disjuncts of an axiom can be completely absorbed, then
Algorithm 4 absorbConcept(C)

Output: Returns the atomic concept for the absorption of C

1: if $C = C_1 \sqcap C_2$ then
2:   $A_1 \leftarrow$ absorbJoined(collectDisjuncts($C_1$, true))
3:   $A_2 \leftarrow$ absorbJoined(collectDisjuncts($C_2$, true))
4:   $T \leftarrow$ fresh atomic concept
5:   $T' \leftarrow T' \cup \{A_1 \sqsubseteq T, A_2 \sqsubseteq T\}$
6:   return $T$
7: else if $C = \forall r.C'$ then
8:   $A_{nb} \leftarrow$ absorbJoined(collectDisjuncts($C'$, true))
9:   $T \leftarrow$ fresh atomic concept
10:  $T' \leftarrow T' \cup \{A_{nb} \sqsubseteq \forall r.T\}$
11: return $T$
12: else if $C = \leq n r.C'$ then
13:  $A_{nb} \leftarrow$ absorbJoined(collectDisjuncts(nnf($\neg C'$), true))
14:  $T \leftarrow$ fresh atomic concept
15:  $T' \leftarrow T' \cup \{A_{nb} \sqsubseteq \forall r.T\}$
16: return $T$
17: else if $C = \neg (a)$ then
18:  $T \leftarrow$ fresh atomic concept
19:  $T' \leftarrow T' \cup \{[a] \sqsubseteq T\}$
20: return $T$
21: else if $C = \neg A$ then
22:   if $A \equiv C' \notin T$ then
23:     return $A$
24:   else
25:     for all $A \equiv C' \in T$ do
26:       $A' \leftarrow$ absorbJoined(collectDisjuncts(nnf($\neg C'$), true))
27:     $T' \leftarrow T' \cup \{A' \sqsubseteq A\}$
28: end for
29: return $A'$
30: end if
31: end if

an empty disjunction is created (line 16 and 20, which corresponds to $\bot$. The methods absorbJoined (Algorithm 3) and absorbConcept (Algorithm 4) are recursively calling each other, whereby absorbJoined is joining several atomic concepts with binary absorption axioms and absorbConcept creates the absorption for a specific, absorbable concept, i.e., absorbJoined handles the absorbable disjunctions, whereas the remaining absorbable concepts are handled by absorbConcept. For instance, a concept of the form $\forall r.C'$ can be absorbed (lines 7–11 of Algorithm 4) by creating a propagation from the atomic concept $A_{nb}$, which is obtained by the absorption of the concept $C'$, back over the $r$-edge, to trigger a fresh atomic concept $T$. Note, if $C'$ cannot be absorbed, then absorbJoined returns $\top$ and the axiom $\top \sqsubseteq \forall r.T$ is created, which corresponds to $\exists r.\top \sqsubseteq T$ and, thus, is similar to the well known role absorption technique [20]. Of course, the absorption can be extended to concept constructors of more expressive DLs.
Algorithm 5 absorbTBox

Output: Creates a new TBox \( T' \) with absorbed axioms for the original TBox \( T \)

1: for all \( X \in T \) do
2: if \( X = A \sqsubseteq C \) then
3: \( T' \leftarrow T' \cup \{A \sqsubseteq C\} \)
4: else if \( X = A \equiv C \) then
5: \( A' \leftarrow \text{absorbJoined}(\text{collectDisjuncts}(\text{nnf}(\neg C), \text{true})) \)
6: if isCA(\text{nnf}(\neg C)) then
7: \( T' \leftarrow T' \cup \{A \sqsubseteq C, A' \sqsubseteq A, A' \subseteq A^+\} \)
8: else if \( \#\{A \sqsubseteq C' \in T \} \cup \{A \equiv C' \in T\} > 1 \) then
9: \( T' \leftarrow T' \cup \{A \sqsubseteq C, A' \sqsubseteq \text{nnf}(\neg C \sqcup A), A' \subseteq A^+\} \)
10: else
11: \( T' \leftarrow T' \cup \{A \equiv C, A' \subseteq A^+\} \)
12: end if
13: else if \( X = C \sqsubseteq D \) or \( X = C \equiv D \) then
14: \( A' \leftarrow \text{absorbJoined}(\text{collectDisjuncts}(\text{nnf}(\neg C \sqcup D), \text{true})) \)
15: \( \{D_1, \ldots, D_n\} \leftarrow \text{collectDisjuncts}(\text{nnf}(\neg C \sqcup D), \text{false}) \)
16: \( T' \leftarrow T' \cup \{A' \sqsubseteq D_1 \sqcup \ldots \sqcup D_n\} \)
17: if \( X = C \equiv D \) then
18: \( A'' \leftarrow \text{absorbJoined}(\text{collectDisjuncts}(\text{nnf}(\neg D \sqcup C), \text{true})) \)
19: \( \{C_1, \ldots, C_m\} \leftarrow \text{collectDisjuncts}(\text{nnf}(\neg D \sqcup C), \text{false}) \)
20: \( T' \leftarrow T' \cup \{A'' \sqsubseteq C_1 \sqcup \ldots \sqcup C_m\} \)
21: end if
22: end if
23: end for

(which we have denoted by “…” in the algorithms), for example, the \( \neg \exists r.\text{Self} \) concept of \( SROIQ \) can be partially absorbed with \( T \sqsubseteq \forall r^{-} A \).

For an atomic concept \( A \), which is completely defined by an axiom of the form \( A \equiv C \) in \( T \), it is often inefficient to decompose the axiom \( A \equiv C \) in \( A \sqsubseteq C \) and \( C \sqsubseteq A \), because \( \text{nnf}(\neg C) \) might not be completely absorbable and then the disjunction \( (\neg C \sqcup A) \) has to be processed for the nodes in the completion graph. In order to determine the satisfiability of a concept, it is, however, for many nodes not relevant whether \( A \) or \( \neg A \) is in their label as long as it can be ensured that one of both alternatives is not causing a clash. Obviously, if the atomic concept \( A \) is only defined once in the knowledge base, then \( C \) or \( \neg C \) and therefore also \( A \) or \( \neg A \) must be satisfiable and only in this case the axiom \( A \equiv C \) can be directly handled by a separate rule, which unfolds \( A \) to \( C \) and \( \neg A \) to \( \neg C \). Thus, there is no need to rewrite the axiom \( A \equiv C \) for an efficient handling in \( T' \) (Algorithm 5, line 11). If there are more definitions for \( A \), then \( A \sqsubseteq C \) as well as \( C \sqsubseteq A \) must be explicitly represented in the new TBox \( T' \) so that possible interactions between these several definitions can be handled. Of course, if \( \text{nnf}(\neg C) \) is partially absorbable, then the disjunction \( \neg C \sqcup A \) can be triggered with \( A' \), which is generated for the absorption of \( \text{nnf}(\neg C) \) (Algorithm 5, line 8).

Furthermore, for each atomic concept \( A \), which is completely defined by an axiom \( A \equiv C \) in \( T \), a candidate concept \( A^+ \) is generated (line 7,9 and 11 of Algorithm 5). An occurrence of \( A^+ \) in the label of a node signalises that the node might be an instance of
the concept \( A \), which is obviously the case if \( A \) itself is in the label, but this is also the case if \( \neg A \) cannot safely be added. As described above, we are, however, not interested in forcing the decision between \( A \) and \( \neg A \) for all nodes in the completion graph. In contrast, we generate the candidate concept \( A' \) that can be used in the absorption instead of \( A \) if \( \neg A \) occurs, whereby it is often possible to delay branching significantly. The creation of \( A' \) is realised by absorbing the concept \( \text{nff}(\neg C) \) for the axiom \( A \equiv C \) for which the disjunction \( \neg C \cup A \) is represented as \( \neg A \sqsubseteq \neg C \). The absorption of \( \text{nff}(\neg C) \) generates the atomic concept \( A' \) and the axiom \( A' \sqsubseteq A'' \) is added to \( T' \) (line 9 of Algorithm 5). If \( \text{nff}(\neg C) \) is not absorbable, then the absorption returns \( \top \) for \( A' \) and \( \top \sqsubseteq A'' \) is added to \( T' \). Note, we also generate the candidate concepts in the \text{absorbConcept} function (lines 25-28 of Algorithm 4) in order to make the absorption of a separate concept complete for the proofs. Of course, if a candidate concept is already created, then it is not necessary to create it again. Besides using candidate concepts in the absorption, they can also be used to identify completely defined concepts as possible subsumers and, therefore, it is almost always useful to generate the candidate concepts [5].

The \text{absorbJoined} function creates binary absorption axioms (Algorithm 3, lines 6-10) for the atomic concepts returned by \text{absorbConcept}. Thus, \text{absorbJoined} is joining several conditions into one fresh atomic concept, which can be used for further absorption or to initiate the addition of the remaining and non-absorbable part of the axiom. In principle, it is not necessary to always create new axioms with fresh atomic concepts for the absorption of identical concepts. In practise, the binary absorption axioms as well as the axioms for absorbing specific concepts can be reused.

\textbf{Example 1.} As an example, the TBox \( T = \{ \exists r.([a] \cap \exists r.[b] \cap \exists r.[c]) \sqsubseteq \exists s.[a] \} \) is rewritten to \( T' \) by the \text{absorbTBox} procedure, where the new axioms in \( T' \) are:

\[
\begin{align*}
[a] & \sqsubseteq T_1 \\
[b] & \sqsubseteq T_2 \\
[c] & \sqsubseteq T_4 \\
T_4 & \sqsubseteq \forall r^{-}.T_5 \\
(T_5 \cap T_6) & \sqsubseteq T_7 \\
T_7 & \sqsubseteq \forall r^{-}.T_8 \\
T_8 & \sqsubseteq \exists s.[a].
\end{align*}
\]

\( T_1, \ldots, T_8 \) are fresh atomic concepts generated by the absorption. To process the one and only axiom \( \exists r.([a] \cap \exists r.[b] \cap \exists r.[c]) \sqsubseteq \exists s.[a] \) in \( T \) the \text{absorbJoined} function is called for the absorbable parts of \( \forall r.(-[a] \cup \forall r.[-[b] \cup \forall r.[-[c]] \cup \exists s.[a], \) which is only the disjunct \( \forall r.(-[a] \cup \forall r.[-[b] \cup \forall r.[-[c]] \). An atomic concept for this disjunct is created with the \text{absorbConcept} function by processing the \( \forall \)-concept and recursively absorbing its qualification \( -[a] \cup \forall r.[-[b] \cup \forall r.[-[c]] \), where the qualification can be simplified into the disjuncts \( -[a], \forall r.[-[b] \) and \( \forall r.[-[c] \). The disjunct \( -[a] \) can be directly absorbed to \([a] \sqsubseteq T_1 \), for the other disjuncts another recursion is necessary to firstly generate \([b] \sqsubseteq T_2 \) and \([c] \sqsubseteq T_4 \) and afterwards the associated propagation of the triggers \( T_3 \) and \( T_5 \) over the \( r^{-} \)-role with \( T_3 \sqsubseteq \forall r^{-}.T_3 \) and \( T_4 \sqsubseteq \forall r^{-}.T_5 \). The \text{absorbJoined} function is joining the atomic concepts \( T_1, T_3 \) and \( T_5 \) by the new binary axioms \( (T_1 \cap T_5) \sqsubseteq T_6 \) and \( (T_3 \cap T_6) \sqsubseteq T_7 \). Now, the absorption of the outer \( \forall \)-concept can be finished by adding the axiom \( T_7 \sqsubseteq \forall r^{-}.T_8 \). The remaining non-absorbable part of the disjunction is handled with \( T_8 \sqsubseteq \exists s.[a]. \) The new axioms are not causing any non-determinism and, furthermore, they can be handled very efficiently as assertions (e.g., \([a] \sqsubseteq C \) is equivalent to the assertion \( C(a) \)), by \textit{lazy unfolding} [1] (for axioms of the form \( A \sqsubseteq C \)) and by a binary absorption rule (for axioms of the form \( (A_1 \cap A_2) \sqsubseteq C \)).
Example 2. For a more complex example, let $\mathcal{T} = \{A_1 \equiv \exists r. A_2 \sqcap \forall r. A_3, (A_4 \sqcup \{a\}) \sqcap \exists s. (A_1 \sqcap \exists r. A_3) \sqsubseteq A_6\}$, which is rewritten to $\mathcal{T}'$ with the axioms:

\[
\begin{align*}
A_2 &\sqsubseteq \forall r^-. T_1 & A_1 &\equiv \exists r. A_2 \sqcap \forall r. A_3 & T_1 &\sqsubseteq A_1^+ \\
\{a\} &\sqsubseteq T_2 & A_4 &\sqsubseteq T_3 & T_2 &\sqsubseteq T_3 \\
A_5 &\sqsubseteq \forall r. T_4 & (A_1^+ \sqcap T_4) &\sqsubseteq T_5 & T_5 &\sqsubseteq \forall s^-. T_6 \\
(T_5 \sqcap T_6) &\sqsubseteq T_7
\end{align*}
\]

First, the axiom $A_1 \equiv \exists r. A_2 \sqcap \forall r. A_3$ is absorbed, which generates the first three axioms of $\mathcal{T}'$, where $A_2 \sqsubseteq \forall r^-. T_1$ and $T_1 \sqsubseteq A_1^+$ are triggering the candidate concept $A_1^+$ of the completely defined concept $A_1$, and the axiom $A_1 \equiv \exists r. A_2 \sqcap \forall r. A_3$ is used for the unfolding of $A_1$. The other eight axioms of $\mathcal{T}'$ are created for the absorption of $(A_4 \sqcup \{a\}) \sqcap \exists s. (A_1 \sqcap \exists r. A_3) \sqsubseteq 3 r.A_3 \sqsubseteq A_6$, similar to the previous example. Again, $T_1, \ldots, T_7$ are the fresh atomic concepts and although not all disjunctions can be eliminated for these axioms, it is nevertheless possible to optimise their structure for a more efficient handling in the tableau algorithm. Additionally to the axioms of the form $\{a\} \sqsubseteq C, A \sqsubseteq C, (A_1 \sqcap A_2) \sqsubseteq C$, which are also created in the previous example, the absorption ensures that all remaining axioms of the form $A \equiv C$ can also be efficiently handled by lazy unfolding, i.e., $A$ can be unfolded to $C$ and $\neg A$ to $\neg C$. Moreover, the new axioms dramatically delay or completely avoid non-deterministic branching caused by disjunctions. For example, without absorption, the disjunction $(\neg A_4 \sqcap \neg\{a\}) \sqcup \forall s. (\neg A_1 \sqcup 2 r.A_4) \sqcup A_5$ obtained from the second axiom in the unprocessed TBox $\mathcal{T}$ has to be processed for each node in a completion graph. With absorption, we have one remaining disjunction, $\forall s. (\neg A_2 \sqcup 2 r.A_3) \sqcup A_4$, which is triggered by $T_7$. The concept $T_7$ is only added to the label of a node if the original disjunction is not trivially satisfiable. For example, if a node does not have any $s$-neighbours, then $T_6$ would not be added to the label and as a consequence of the axiom $(T_5 \sqcap T_6) \sqsubseteq T_7$, $T_7$ would also not be added. In this case, it would not be necessary to make non-deterministic decisions since the second disjunct $\forall s. (\neg A_1 \sqcup 2 r.A_3)$ of the original disjunction is trivially satisfiable. Please also note that the decision between $\neg A_1$ and $A_1$ is not enforced for every node in a completion graph for $\mathcal{T}'$ and, nevertheless, $A_1$ can be partially used in the absorption by replacing it with $A_1^+$. As long $A_1^+$ is not in the label of a node of a fully expanded completion graph, we know that $\neg A_1$ must be satisfiable, because $\forall r. \neg A_2$ of the disjunction $\forall r. \neg A_2 \sqcup \exists r. \neg A_3$ cannot not cause a clash. This can be utilised in the absorption of the other axiom, because as long as we know that $\neg A_1$ can be added to a $s$-neighbour without causing a clash, we also know that the axiom $(A_4 \sqcup \{a\}) \sqcap \exists s. (A_1 \sqcap 3 r.A_3) \sqsubseteq A_6$ can be trivially satisfied.

3.1 Correctness

Termination for the absorption algorithm itself is ensured by the acyclicity of the axioms. However, if it is ensured that the candidate concepts are exclusively created by the absorbTBox procedure for all atomic concepts, which are completely defined with axioms of the form $A \equiv C$ in $\mathcal{T}$, then in the recursion between absorbJoined and absorbConcept only the current axiom has to be processed. Since we generate the
candidate concepts in absorbConcept only to make the functions absorbJoined and absorbConcept complete for the absorption of a concept for the proofs, the acyclicity restriction would not be necessary for termination.

In the following we prove the correctness of our modified absorption algorithm. We first show that the complete absorption of a disjunct of an axiom is correct, i.e., it preserves the satisfiability (Lemma 1 and Lemma 2), and then we show that the correctness of a partially absorbed concept disjunct can be reduced to the complete absorption (Lemma 3).

Lemma 1  Let \( T \) denote a TBox, \( I = (\Delta^I, \cdot^I) \) an interpretation such that \( I \models T \), \( C \) a concept that is completely absorbable, \( A \) the concept returned by absorbJoined(\( \{C\} \)), and \( T' \) the extension of \( T \) with all the axioms created by absorbJoined(\( \{C\} \)), then

1. for every extension \( I' \) of \( I \) such that \( I' \models T' \), it holds that \( I' \models T \),
2. for every extension \( I' \) of \( I \) such that \( I' \models T' \), it holds for all \( \delta \in \Delta^F \) that \( \delta \in \Delta^F \)
   if \( \delta \notin C^F \), and
3. there exists an interpretation \( I' = (\Delta^I, \cdot^I) \) such that \( I' \models T' \) with \( \Delta^F = \Delta^I \) and \( \delta \in \Delta^F \) only if \( \delta \notin C^F \).

Proof. (Claim 1) Since \( T' \) is an extension of \( T \), it trivially follows that \( I' \models T \).

(Claim 2) We first prove the simple cases where \( C \) is completely absorbable and afterwards we show by induction that the lemma also holds for the complex cases.

- If \( C \) is of the form \( \neg A \) and \( A \equiv C' \notin T \), then absorbConcept(\( \{C\} \)) directly returns \( A \), which is then also returned by absorbJoined(\( \{C\} \)). Thus, if \( \delta \in \Delta^F \) and \( \delta \notin C^F \), i.e., \( \delta \notin (\neg A)^F \), then \( \delta \in \Delta^F \). Hence, the lemma holds if \( C \) is of the form \( \neg A \).
- If \( C \) is of the form \( \neg \{a\} \), then absorbConcept(\( \{C\} \)) adds the axiom \( \{a\} \subseteq A \) to \( T' \) and returns \( A \), which is then also returned by absorbJoined(\( \{C\} \)). Thus, if \( \delta \notin C^F \), i.e., \( \delta \notin (\neg \{a\})^F \), then \( \delta \in \Delta^F \) and because, by assumption, \( I' \models T' \), i.e., \( I' \models \{a\} \subseteq A \), it follows that \( \delta \in \Delta^F \). Hence, the lemma holds if \( C \) is of the form \( \neg \{a\} \).

For the complex cases we assume that all nested disjunctions are replaced by a single disjunction with all disjuncts, i.e., \( (C_1 \sqcup (C_2 \sqcup C_3)) \) is replaced by \( (C_1 \sqcup C_2 \sqcup C_3) \). Furthermore, we automatically decompose a disjunction into the set of disjuncts by calling absorbJoined. This simplification is also done by the algorithm with the collectDisjuncts function, which is always called before absorbJoined. Therefore, we can omit collectDisjuncts for calling absorbJoined, which improves the readability.

Now, for a disjunct \( C_j \), it follows that \( C_j \) is not a disjunction itself and it also follows that absorbJoined(\( \{C_j\} \)) only returns the atomic concept that is returned by absorbConcept(\( C_j \)).

Let \( C_1, \ldots, C_n \) be completely absorbable concepts and \( A_1, \ldots, A_n \) the atomic concepts returned by absorbJoined(\( \{C_1\} \), \ldots, absorbJoined(\( \{C_n\} \)). By our induction hypothesis, the lemma holds for \( A_1 \) w.r.t. \( C_1, \ldots, A_n \) w.r.t. \( C_n \).

- If \( C \) now of the form \( C_1 \sqcup \ldots \sqcup C_n \), then absorbJoined(\( \{C_1, \ldots, C_n\} \)) collects the atomic concepts \( A_1, \ldots, A_n \) by calling absorbConcept(\( C_j \)) for each \( C_j, 1 < j < n \), and creates the binary absorption axioms \( (A_1 \cap A_2) \subseteq T_1, (T_1 \cap A_3) \subseteq T_2, \ldots, (T_{n-2} \cap A_n) \subseteq A \). Thus, if \( \delta \notin C^F \), i.e., \( \delta \notin (C_1 \sqcup \ldots \sqcup C_n)^F \), then \( \delta \in (\neg C_1 \sqcup \ldots \sqcup \neg C_n)^F \) and as a consequence \( \delta \in (\neg C_j)^F \) for \( 1 < j < n \). Therefore, by the induction
hypothesis we have $\delta \in A_j^T$ for all $1 \leq j \leq n$. Thus, $\delta \in A_j^T$ and $\delta \in A_i^T$ and since
the interpretation $I' \models T'$ with $\{(A_1 \cap A_2) \subseteq T_1, (T_1 \cap A_3) \subseteq T_2, \ldots, (T_{n-2} \cap A_n) \subseteq A_i \subseteq T_i \}$ it follows that $\delta \in T_i^1$, $\delta \in T_i^2$,..., $\delta \in A_i^T$. Hence, the lemma holds by
induction if $C$ is of the form $C_1 \cap \ldots \cap C_n$.

- If $C$ is of the form $C_1 \cap C_2$, then $\text{absorbConcept}(\langle C \rangle)$ returns $A_i$, which is obtained
by calling $\text{absorbConcept}(C)$, where additionally the axioms $A_1 \subseteq A$ and $A_2 \subseteq A$ are
created. If $\delta \notin C_i$, i.e., $\delta \notin (C_1 \cap C_2)^T$, then $\delta \in (\neg C_1 \cup \neg C_2)^T$. There are
now two cases: If $\delta \in (\neg C_1)^T$, then by the induction hypothesis we have $\delta \in A_i^T$ and
due to the axiom $A_1 \subseteq A$ we have $\delta \in A_i^T$. For the other case we have $\delta \in (\neg C_2)^T$
and by the induction hypothesis $\delta \in A_i^T$ and due to the axiom $A_2 \subseteq A$ we also have
$\delta \in A_i^T$. Hence, the lemma holds by induction if $C$ is of the form $C_1 \cap C_2$.

- If $C$ is of the form $\forall r. C_1$, then $\text{absorbConcept}(C)$ creates $A_1 \subseteq \forall r_{-} A$ and $A_i$ is
returned by $\text{absorbConcept}(\langle C \rangle)$. Thus, if $\delta \notin C_i^\tau$, i.e., $\delta \notin (\forall r.C_1)^T$, then $\delta 
\in (\exists r. \neg C_1)^T$. It follows that there exists $\gamma \in A_i^\tau$, $\delta, \gamma \in r^T$ with $\gamma \in (\neg C_i)^T$ and
by the induction hypothesis we have $\gamma \in A_i^T$. As a consequence of the axiom
$A_1 \subseteq \forall r_{-} A$ we also have $\delta \in A_i^T$. Hence, the lemma holds by induction if $C$ is of
the form $\forall r.C_1$.

- If $C$ is of the form $\neg A'$ and $A'$ is completely defined by the axioms $A' \equiv C'_1 \in T, \ldots, A' \equiv C'_n \in T$, then the candidate concept $A'^* \equiv C'_1$, i.e., $C_1, \ldots, C_n$.

Now, for $1 \leq i \leq m$ and for each axiom $H \subseteq A_i$, generated by the absorption, we
exhaustively add $\delta \in A_i^T$ to $A_i^T$ if (i) $H = A'$ and $\delta \in A_i^T$, (ii) $H = \{a\}$ and $\delta \in \{a\}^T$, (iii) $H = (A' \cap A'')$ and $\delta \in A_i^T \cap A''^T$, or (iv) $H = \exists r_{-} A'$ and $\delta \in \exists r_{-} A'_{-}^T$, i.e.,
$\delta$ has some $r$-neighbour $\gamma$ such that $(\gamma, \delta) \in r^T$ and $\gamma \in A_i^T$. We have $\delta \in A_i^T$ only
if $\delta$ satisfies the left-hand side of an axiom $A' \subseteq A_i$, $\{a\} \subseteq A_i$, or $(A' \cap A'') \subseteq A_i$, or
$\exists r_{-} A' \subseteq A_i$. Consequently, it follows that $I' \models T'$. Furthermore, $\delta \notin A_i^T$ if $\delta \in C_i^T$, because of the following cases:

- If $C$ is of the form $\exists A, A \equiv C' \notin T$ and $\delta \in C_i^T$, i.e., $\delta \in (\neg A)^T$, then $\delta \notin A_i^T$.

- If $C$ is of the form $\neg \{a\}$ for which the absorption has generated $\{a\} \subseteq A$ and if
$\delta \in C_i^T$, i.e., $\delta \in (\neg \{a\})^T$, then $\delta \notin \{a\}^T$ and then $\delta \notin A_i^T$, because the left-hand
side of \([a] \subseteq A\) is not satisfied and there is also no other axiom that implies \(A\), because \(A\) is freshly used for \([a] \subseteq A\).

For the remaining cases, we again assume that the lemma holds for \(A_1\) w.r.t. \(C_1, \ldots, C_n\), where \(A_1, \ldots, A_n\) are the atomic concepts for absorbing the completely absorbable concepts \(C_1, \ldots, C_n\). Therefore, it follows by induction that \(\delta \not\subseteq A^T\) if \(\delta \in C^T\), because:

- If \(C\) is of the form \(C_1 \sqcup \ldots \sqcup C_n\) and \(\delta \in C^T\), i.e., \(\delta \in (C_1 \sqcup \ldots \sqcup C_n)^T\), then there exists a \(C_j\) \(1 \leq j \leq n\) with \(\delta \in C_j^T\). By the induction hypothesis it follows that \(\delta \not\subseteq A_j^T\) and by the binary axiom chain \((A_1 \sqcap A_2) \subseteq T_1, (T_1 \sqcap A_3) \subseteq T_2, \ldots, (T_{j-1} \sqcap A_j) \subseteq T_{j-1}, \ldots, (T_{n-2} \sqcap A_n) \subseteq A\), which is generated for absorbing \(C_1 \sqcup \ldots \sqcup C_n\), we have \(\delta \not\subseteq A^T\), because the left-hand side of the axiom \((T_{j-2} \sqcap A_j) \subseteq T_{j-1}\) cannot be satisfied.

- If \(C\) is of the form \(C_1 \sqcap C_2\) and \(\delta \in C^T\), i.e., \(\delta \in (C_1 \sqcap C_2)^T\), then \(\delta \in C_1^T\) and \(\delta \in C_2^T\). By the induction hypothesis we have \(\delta \not\subseteq A_1^T\) and \(\delta \not\subseteq A_2^T\). The left-hand side of the axioms \(A_1 \subseteq A\) and \(A_2 \subseteq A\) is not satisfied and the absorptions does not generate other axioms that imply \(A\). Thus, \(\delta\) is not added to \(A^T\).

- If \(C\) is of the form \(\forall r.C_1\) and \(\delta \in C^T\), i.e., \(\delta \in (\forall r.C_1)^T\), then for all \(\gamma \in A^T\) with \((\delta, \gamma) \not\in r^T\) we also have \(\gamma \not\subseteq C_1^T\). By the induction hypothesis it follows that \(\gamma \not\subseteq A_1^T\) and since the left-hand side of the generated axiom \(\forall r^+\cdot A_1 \subseteq A_1\) is not satisfied, and there are not any other axioms that imply \(A\), we do not add \(\delta\) to \(A^T\) and, thus, \(\delta \not\subseteq A^T\).

- If \(C\) is of the form \(\neg \cdot A'\). \(A'\) is completely defined by the axioms \(A' \equiv C_1' \in T, \ldots, A' \equiv C_n' \in T\), and \(\delta \in C^T\), i.e., \(\delta \in (\neg \cdot A')^T\), then for \(C_1 = \neg C_1', \ldots, C_n = \neg C_n'\) and as a consequence of the axioms \(A' \equiv C_1', \ldots, A' \equiv C_n'\), we also have \(\delta \not\subseteq C_1^T, \ldots, \delta \not\subseteq C_n^T\). By the induction hypothesis we have \(\delta \not\subseteq A_1^T, \ldots, \delta \not\subseteq A_n^T\) and, thus, the left-hand side of all axioms that imply the candidate concept \(A = A'^\ast\) is not satisfied. Therefore, we have \(\delta \not\subseteq A^T\).

We can now use Lemma 1 to show the correctness of the absorption for the case of a completely absorbable concept \(C\) in an axiom \(C \subseteq D\).

**Lemma 2** For \(T\) a TBox and \(C \sqcup D\) a disjunction, where \(C\) is completely absorbable and \(D\) is neither completely nor partially absorbable, let \(T_1\) denote the TBox with \(T_1 = T \sqcup (T \subseteq C \cup D)\) and \(T_2\) denote the TBox with \(T_2 = T \sqcup (A \subseteq D) \cup X\), where \(X\) are the axioms created by \(A \leftarrow \text{absorbJoined}(C)\). Then, a concept \(C'\) is satisfiable with respect to \(T_1\) if it is satisfiable with respect to \(T_2\).

**Proof.** If direction: For \(I_2\) an interpretation with \(C^T_2 \neq \emptyset\) and \(I_2 \models T_2\), we show that \(I_2 \models T_1\). Because of the axiom \(A \subseteq D \in T_2\) for each \(\delta \in A^T_2\) it holds that either \(\delta \not\subseteq A^T_2\) (and thus \(\delta \in C^T_2\) by Lemma 1) or \(\delta \in D^T_2\). Thus, the axiom \(T \subseteq (C \cup D) \in T_1\) is satisfied for every \(\delta \in A^T_2\) and, therefore, \(I_2 \models T_1\).

Only if direction: For \(I_1\) an interpretation with \(C^T_1 \neq \emptyset\) and \(I_1 \models T_1\), we construct an interpretation \(I'_1\) with \(C^T_1 \neq \emptyset\) and \(I'_1 \models T_2\). Since \(I_1 \models T_1\) and \(T_1\) is an extension of \(T\), it follows that \(I'_1 \models T\). Because of Lemma 1, there exists an interpretation \(I'\) that can be constructed from \(I_1\) for which it holds that \(I'_1 \models T \cup X\) and for all \(\delta \in A^T_1\)
that $\delta \in A^I$ only if $\delta \notin C^I$. Thus, it also follows that $I'_1 \models A \sqsubseteq D$, because $A^I_1 = A^I$, and for all $\delta \in A^I$ it holds that either $\delta \in C^I_1$ and thus $\delta \notin A^I$ or $\delta \notin D^I$. Thus, if $C^I_1 \neq \emptyset$, then $C^I_1 \neq \emptyset$.

In order to show the correctness of the partial absorption of a disjunction $C \sqcup D$, where $C$ is partially absorbable and $D$ is neither completely nor partially absorbable, we reduce the problem to the complete absorption of $C' \sqcup C \sqcup D$, where for $C'$ it holds that $C'$ is completely absorbable and $C' \subseteq C$. We show that the partial absorption of $C$ is equivalent to the complete absorption of the concept $C'$. Therefore, the partial absorption of $C \sqcup D$ corresponds to the complete absorption of $C' \sqcup C \sqcup D$, which is obviously equisatisfiable to $C \sqcup D$ since $C$ subsumes $C'$.

**Lemma 3** Let $C$ be a partially absorbable concept, then $\text{absorbJoined}([C])$ generates the absorption of a concept $C'$ for which it holds that $C' \subseteq C$ and $C'$ is completely absorbable.

**Proof.** If $C$ is already completely absorbable, then the lemma trivially holds since in this case $C'$ is $C$. Thus, we show in the following for all cases where $C$ is partially absorbable but not completely absorbable that $\text{absorbJoined}([C])$ generates the absorption of a more specific concept $C'$ for which it holds $C' \subseteq C$ and $C'$ is completely absorbable.

- If $C$ is of the form $\forall r.D'$ (of the form $\leq n \ r. D'$ with $n \geq 0$) and $D'$ ($\neg \text{nnf}(\neg D')$) is neither completely nor partially absorbable, then $\text{absorbConcept}(C)$ creates $\top \sqsubseteq \forall r.\neg A$, which corresponds to the complete absorption of $\forall r.\neg \top$ for which it holds that $\forall r.\neg \top \sqsubseteq \forall r.D'$ ($\forall r.\neg \top \sqsubseteq \forall r. D'$ for $n \geq 0$).

To prove the complex cases by induction, we assume that the concepts $D_1, \ldots, D_m$ are partially absorbable and the lemma holds for $D_1, \ldots, D_m$, i.e., the absorption completely absorbs the concepts $D_1', \ldots, D_m'$, for which it holds that $D_1' \sqsubseteq D_1, \ldots, D_m' \sqsubseteq D_m$, and let $A_1, \ldots, A_m$ be the atomic concepts that are achieved for absorbing $D_1', \ldots, D_m'$.

- If $C$ is of the form $\forall r.D_1$ (of the form $\leq n \ r. D'$ with $n \geq 0$ and $D_1 = \neg \text{nnf}(\neg D')$) and $D_1$ is partially absorbable, then $\text{absorbConcept}(C)$ creates $A_1 \sqsubseteq \forall r.\neg A$, where $A_1$ is the atomic concept that is returned by $\text{absorbConcept}(D_1)$ for completely absorbable $D_1'$. The absorption of $C$ corresponds to the complete absorption of $\forall r. D_1'$ and, by the induction hypothesis, we have $D_1' \sqsubseteq D_1$. Thus, it also holds that $\forall r.D_1' \sqsubseteq \forall r.D_1$ ($\forall r.D_1' \sqsubseteq \forall r. D'$ for $n \geq 0$, because $D_1' \sqsubseteq \neg D'$).

- If $C$ is of the form $D_1 \sqcup \ldots \sqcup D_m \sqcup C_1 \sqcup \ldots \sqcup C_p$ with $D_1, \ldots, D_m$ partially absorbable and $C_1, \ldots, C_p$ neither partially nor completely absorbable, then the absorption creates the binary axiom chain $(A_1 \sqcap A_2) \sqsubseteq T_1, (T_1 \sqcap A_3) \sqsubseteq T_2, \ldots, (T_{m-2} \sqcap A_m) \sqsubseteq A$, which corresponds to the complete absorption of $D_1' \sqcup \ldots \sqcup D_m'$, where $A_1, \ldots, A_m$ are again the atomic concepts for absorbing $D_1', \ldots, D_m'$. Because of the induction hypothesis it holds that $D_1' \sqcup \ldots \sqcup D_m' \sqsubseteq D_1 \sqcup \ldots \sqcup D_m \sqcup C_1 \sqcup \ldots \sqcup C_p$.

- If $C$ is of the form $D_1 \sqcap D_2$ with $D_1, D_2$ partially absorbable, then the absorption creates the axioms $A_1 \sqsubseteq A$ and $A_2 \sqsubseteq A$, which corresponds to the complete absorption of $D_1' \sqcap D_2'$, where $A_1$ and $A_2$ are the atomic concepts for absorbing $D_1'$ and $D_2'$. Because of the induction hypothesis it holds that $D_1' \sqcap D_2' \sqsubseteq D_1 \sqcap D_2$.
• If \( C \) is of the form \( \neg A \), \( A \) is completely defined by the axioms \( A \equiv \neg \hat{D}_1 \in \mathcal{T}, \ldots, A \equiv \neg \hat{D}_m \in \mathcal{T} \) and \( D_1 = \neg \hat{D}_1, \ldots, D_m = \neg \hat{D}_m \) are partially absorbable, then the absorption returns \( A^+ \) and also creates the axioms that imply the candidate concept \( A^+ \). Let \( A' \) be the atomic concept that is completely defined by the axioms \( A' \equiv \neg \hat{D}'_1, \ldots, A' \equiv \neg \hat{D}'_m \), then the complete absorption of \( D'_1, \ldots, D'_m \) creates the atomic concepts \( A_1, \ldots, A_m \) and additionally the axioms that imply \( A'^+ \). The partial absorption of \( \neg A \) corresponds to complete absorption of \( \neg A' \) and thus it obviously holds by the induction hypothesis that \( \neg A' \subseteq \neg A \) and \( A'^+ \subseteq A^+ \).

As a consequence of the above lemmas, we find that the above presented absorption algorithms indeed produce a TBox for which concept satisfiability is preserved:

**Theorem 1** Let \( \mathcal{T} \) denote a TBox, which is rewritten into \( \mathcal{T}' \) by the absorbTBox function, and \( C \) a concept, then \( C \) is satisfiable with respect to \( \mathcal{T} \) iff it is satisfiable with respect to \( \mathcal{T}' \).

## 4 Nominal Schema Absorption

Axioms with nominal schemas are very expressive in comparison to many decidable alternatives based on rules. For instance, the atoms in the heads or bodies of DL-safe SWRL rules can only be instantiated with individuals that occur in the ABox. In tableau algorithms, it is, therefore, only necessary to check whether the bodies of such rules are satisfied on tableau nodes that represent ABox individuals/nominals. If this is the case, then the atoms of the heads have to be added, however, also exclusively to individual/nominal nodes. This is no longer the case for axioms with nominal schemas. For example, given the nominal schema axiom

\[
\exists r. (\exists r. (\exists x. (x) \cap \exists x. (x)) \subseteq \exists r. (x),
\]

we have to check whether the left-hand side is satisfied at any tableau node, although the variable \( x \) can only bind to nodes that represent individuals/nominals. That is, checking may also involve blockable nodes in the completion graph that do not represent ABox individuals. Furthermore, such axioms can then enforce the addition of the right-hand side also on blockable nodes. As a consequence, typical approaches for rule processing, such as Rete [4], cannot be used in a straightforward way since blocking easily becomes unsound.

Due to the fact that it would be necessary to also process all blockable nodes with the Rete algorithm, i.e., also the concepts in the label of blockable nodes as well as the roles in the edge labels to these blockable nodes have to be used as input facts for the Rete algorithm, and because Rete does not provide blocking information, it is not clear when the expansion of new successors can be stopped in the tableau algorithm. For example, if the knowledge base also contains the axiom \( \top \subseteq \exists r. \top \cap \exists r. \{a\} \cap \exists x. \{a\} \) and the construction of new successors is already blocked after, e.g., two successively created \( t \)-successors, then the Rete algorithm cannot infer the right-hand side of the considered nominal schema axiom for any constructed node since this requires the successive creation of at least three \( t \)-successors. Obviously, this is even more complicated if some of the roles are complex.
Our approach to overcome this issue is to emulate well known rule processing algorithms such as Rete by adapted tableau rules, which propagate bindings of variables for concepts through the completion graph. The propagated bindings of variables can be considered in the blocking condition, which allows for ensuring completeness, soundness and termination. As a nice side-effect, the propagation of bindings in the completion graph also means that complex roles can be supported without further adjustments.

This approach works well if the axioms have a typical rule structure, i.e., the axioms have a large absorbable part and almost every nominal schema variable appears at least once in the absorbable part. This is hardly surprising, because ordinary GCIs without nominal schema variables must also have a large absorbable part for reasoning systems to handle such axioms efficiently.

In order to actually bind variables to individuals (or nodes in a completion graph), we use the ↓ binder operator, as known from Hybrid Logics [2]. The unrestricted extension of a Description Logic with binders easily leads to undecidability of the standard reasoning problems. However, we retain the decidability since we only bind variables to individuals that occur in the ABox. In order to realise this, we extend a knowledge base with nominal schemas with axioms of the form \([a] \sqsubseteq O\) for each individual \(a\), where \(O\) is a fresh atomic concept, and the axioms created by the absorption ensures that binders are then only triggered in the completion graph if the special concept \(O\) occurs in the label of the node. In the remainder of this paper, we assume that all considered knowledge bases already contain the \([a] \sqsubseteq O\) axioms for each ABox individual \(a\).

### 4.1 Absorption of Axioms with Nominal Schemas

The absorption of axioms with nominal schema variables works very similar to the absorption of ordinary axioms without nominal schema variables. Typically, the absorption algorithm can be directly extended to handle the new concept construct. However, to avoid some special cases for conjunctions \(C_1 \sqcap C_2\) in an absorbable disjunct, where different nominal schema variables are used in \(C_1\) and \(C_2\), we require for the nominal schema absorption that all conjunctions in absorbable positions are eliminated. This can be done by duplicating the disjunction that is absorbed and replacing the corresponding conjunction in one case with \(C_1\) and in the other case with \(C_2\). For example, the axiom \([x] \sqcup A \sqsubseteq \exists r. [x]\) is handled as the disjunction \((\neg [x] \sqcap \neg A) \sqcup \exists r. [x]\) in the absorption and the conjunction \(\neg [x] \sqcap \neg A\) has to be eliminated by replacing the original axiom with \([x] \sqsubseteq \exists r. [x]\) and \(A \sqsubseteq \exists r. [x]\). For our absorption algorithm of Section 3, the following two modifications are necessary in order to handle nominal schemas in the remaining axioms (cf. Algorithms 6 and 7):

- **isCA(C) (isPA(C))** is extended to return that a negated occurrence of a nominal schema \(\neg[x]\) is completely (partially) absorbable.
- **absorbConcept(C)** must now also handle a negated occurrence of a nominal schema \(\neg[x]\) by absorbing it to \(O \sqsubseteq \downarrow x. T_x\), where \(T_x\) is a fresh atomic concept and \(O\) is the special atomic concept that is added to the label of every individual \([a]\) in the ABox by axioms of the form \([a] \sqsubseteq O\).
Algorithm 6 Absorption extensions for Algorithms 1

1: procedure isCA(C) 

... 

2: if C = ¬{x} then 
3: return true 
4: end if 

... 

5: end procedure 

Algorithm 7 Absorption extensions for Algorithm 4

1: procedure absorbConcept(C) 

... 

2: if C = ¬{x} then 
3: T_x ← fresh atomic concept 
4: T' ← T' ∪ {O ⊑↓ x. T_x} 
5: return T_x 
6: end if 

... 

7: end procedure 

Other concepts can be absorbed as before, however, the final atomic concept A created by the absorption cannot initiate the addition of the remaining, non-absorbed part of the axiom in the same way. If the remaining disjuncts D_1, ..., D_n still contain nominal schemas, then the disjunction has to be grounded with those bindings of variables that have been propagated to A. In the tableau algorithm this can be done dynamically, e.g., with a new “grounding concept” and a corresponding rule. Therefore, if D_1, ..., D_n still contain concepts with nominal schema variables, then A ⊑ gr(D_1 ∪ ... ∪ D_n) has to be added to the TBox, where gr(C) is the new grounding concept. For simplicity, let us assume that gr(C) is always used to add the remaining, non-absorbed part of the axiom, even if C or the axiom does not contain any nominal schema variables.

Example 3. As an example, the axiom ∃c.(∃x.R(x) ∩ ∃a.(∃y.R(y) ∩ ∃v.S(z)∆)) can be almost completely absorbed into the following axioms:

\[
\begin{align*}
O \in \downarrow x.T_x & \quad O \in \downarrow y.T_y & \quad T_1 \in \forall a^{-}.T_1 \\
O \in \downarrow z.T_z & \quad T_2 \in \forall v^{-}.T_2 & \quad (T_1 \cap T_2) \subseteq T_3 \\
T_3 \in \forall s^{-}.T_4 & \quad (T_3 \cap T_4) \subseteq T_5 & \quad T_3 \in \forall r^{-}.T_6 \\
(T_3 \cap T_5) \subseteq T_7 & \quad T_7 \subseteq gr(∃c.(x)) \\
\end{align*}
\]

Again, T_x, T_y, T_z, T_1, ..., T_7 are fresh atomic concepts. Only ∃c.(x) cannot be absorbed and has to be grounded on demand. In the example, we have reused axioms for the absorption of the same concepts to reduce the total number of axioms. The basic algorithm of Section 3 would generate for each occurrence of ¬{y} a separate binder concept, i.e., we would have \( O \subseteq \downarrow y.T_y \) as well as \( O \subseteq \downarrow y.T'_y \), which is obviously not necessary.
4.2 Tableau Algorithm Extensions to Handle Variable Bindings

We can now extend a standard tableau decision procedure to support (absorbed) nominal schema axioms. The ↓ and gr(-) concepts are handled by new rules. Furthermore, the $\sqsubseteq_1$- and $\sqsubseteq_2$-rules to handle TBox axioms and the $\forall$-rule (for transitivity support also the $\forall^+$-rule) have to be adapted in order to propagate variables bindings.

Roughly speaking, for each concept $C$ in the label of a node $v$, we keep a set of mappings that records bindings for variables. A mapping set is created, when a concept of the form $\exists x . C$ occurs in the label of a node $v$. In this case, we add $C$ to the label of $v$ and, in order to “remember” the binding $x \mapsto v$, we add the mapping $\mu$ with $\mu(x) = v$ to the mappings of $C$. Note that, as a consequence of our absorption algorithm, a binder concept $\exists x . C$ is always such that $C$ does not contain further binders.

**Definition 7 (Variable Mapping).** A variable mapping $\mu$ is a (partial) function from variable names to individual names. For a variable mapping $\mu$ – and more generally for any (partial) function – the set of elements on which $\mu$ is defined is the domain, written $\text{dom}(\mu)$, of $\mu$, and the set $\text{ran}(\mu) = \{ \mu(x) \mid x \in \text{dom}(\mu) \}$ is the range of $\mu$. We use $\epsilon$ for the empty variable mapping, i.e., $\text{dom}(\epsilon) = \emptyset$, and we associate a concept fact $C(v)$ with a set of variable mappings, denoted by $\mathcal{B}(C, v)$.

If no confusing is likely to arise, we simply write mapping instead of variable mapping. The mappings for a concept fact have to be propagated by the tableau rules for the concepts and axioms that are used in the absorption. For example, if we apply the $\sqsubseteq_1$-rule (cf. Table 2) to an axiom of the form $A \sqsubseteq C$, we keep the mappings also for the concept $C$. Similarly, we extend other rules (see Table 2) and we describe the not so straightforward extensions in more detail below. Note, it is only necessary to extend those rules, which are related to concepts and axioms that are used in the absorption, because if the mappings are propagated to the $gr$-concept, then the remaining, non-absorbed part of the axiom is grounded and thus corresponds to an ordinary concept.

Some major adjustments are necessary in order to handle binary absorption axioms of the form $(A_1 \sqcap A_2) \sqsubseteq C$ correctly (cf. $\sqsubseteq_2$-rule). First of all, we want to keep the default behaviour if there are no variable mappings associated to the concept facts for which the rule is applied, i.e., if $\mathcal{B}(A_1, v) \cup \mathcal{B}(A_2, v) = \emptyset$, then we add $C$ to the label of $v$. In contrast, if $\mathcal{B}(A_1, v) \neq \emptyset$ or $\mathcal{B}(A_2, v) \neq \emptyset$, we propagate the join of the mapping sets to the implied concept. In the case $\mathcal{B}(A_1, v) = \emptyset$ and $\mathcal{B}(A_2, v) \neq \emptyset$, we extend $\mathcal{B}(A_1, v)$ by the empty mapping $\epsilon$ so that the join of $\mathcal{B}(A_1, v)$ and $\mathcal{B}(A_2, v)$ results in $\mathcal{B}(A_2, v)$, which is then propagated to $C$. We proceed analogously for $\mathcal{B}(A_2, v) = \emptyset$ and $\mathcal{B}(A_1, v) \neq \emptyset$. In principle, the join combines variable mappings that map common variables to the same individual name and to point out that the empty sets of mappings are specially handled, we have extended the join operator $\otimes$ with the superscript $\epsilon$.

**Definition 8 (Variable Mapping Join).** Two variable mappings $\mu_1$ and $\mu_2$ are compatible if $\mu_1(x) = \mu_2(x)$ for all $x \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$. A variable mapping $\mu_1 \cup \mu_2$ is defined by setting $(\mu_1 \cup \mu_2)(x) = \mu_1(x)$ if $x \in \text{dom}(\mu_1)$, and $(\mu_1 \cup \mu_2)(x) = \mu_2(x)$ otherwise. Given two (possibly empty) sets of variable mappings $M_1, M_2$, let $M^\epsilon_1 = \{ \epsilon \}$ if $M_1 = \emptyset$ and $M^\epsilon_2 = \emptyset$ and $M^\epsilon_2 = M_1 \cap \{ M^\epsilon_2 = M_2 \}$ otherwise. The join $M_1 \otimes M_2$ is defined as $\{ \mu_1 \cup \mu_2 \mid \mu_1 \in M^\epsilon_1, \mu_2 \in M^\epsilon_2 \text{ and } \mu_1 \text{ is compatible with } \mu_2 \} \setminus \{ \epsilon \}$. 
C variable set Y by the empty mapping ϵ and tableau rules as well, whereas the dynamic extension with Let µ schema variables that syntactically occur in C. A concept C is Definition 9 (Grounding, Completion).

Table 2. Tableau rule extensions to propagate variable mappings

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>V-rule:</td>
<td>if ∀r.C ∈ L(v), v not indirectly blocked, there is an r-neighbour w of v with C ∉ L(w) or B(Vr.C, v) ∉ B(C, w), then L(w) → L(w) ∪ [C] and B(C, w) → B(C, w) ∪ B(Vr.C, v)</td>
<td></td>
</tr>
<tr>
<td>≡1-rule:</td>
<td>if A ⊆ C ∈ K, A ⊆ L(v), v not indirectly blocked, and C ∉ L(v) or B(A, v) ∉ B(C, v), then L(v) → L(v) ∪ [C] and B(C, v) → B(C, v) ∪ B(A, v)</td>
<td></td>
</tr>
<tr>
<td>≡2-rule:</td>
<td>if (A1 ∩ A2) ⊆ C ∈ K, [A1, A2] ⊆ L(v), v not indirectly blocked, and 1. B(A1, v) ∪ B(A2, v) = ∅ and C ∉ L(v), or 2. (B(A1, v) ⊑ B(A2, v)) ≠ ∅ and C ∉ L(v) or (B(A1, v) ⊑ B(A2, v)) ∉ B(C, v), then L(v) → L(v) ∪ [C] and B(C, v) → B(C, v) ∪ (B(A1, v) ∪ B(A2, v))</td>
<td></td>
</tr>
<tr>
<td>↓-rule:</td>
<td>if ↓.C ∈ L(v), v not indirectly blocked, and C ∉ L(v) or (x ↦→ v) ∉ B(C, v), then L(v) → L(v) ∪ [C] and B(C, v) → ↓{v ↦→ v}</td>
<td></td>
</tr>
<tr>
<td>gr-rule:</td>
<td>if gr(C) ∈ L(v), v not indirectly blocked, there exists a variable mapping µ ∈ comp_{Vars(C)}^K(B(gr(C), v)) with C[µ] ∉ L(v), then L(v) → L(v) ∪ [C[µ]]</td>
<td></td>
</tr>
</tbody>
</table>

Note, the extension by the empty variable mapping ϵ is required to propagate variable mappings to a concept C if B(A1, v) = ∅ or B(A2, v) = ∅. We cannot simply associate all concept facts also with the empty variable mapping, because all mappings are compatible with the empty variable mapping. Hence, if B(A1, v) ≠ ∅ and B(A2, v) ≠ ∅, also the variable mappings of B(A1, v) and B(A2, v) would directly be propagated to C, whereas only the combination of the mappings should be propagated. Clearly, it would be possible to only associate all concept facts in the initial completion graph with the empty variable mapping. One could then propagate these empty variable mappings to newly added concepts as long as no other variable mappings are created for these concepts through the binder rule ↓-rule. However, it would be necessary to adapt all other tableau rules as well, whereas the dynamic extension with ϵ during a join allows for not modifying other tableau rules.

Besides the new ↓-rule, we also have to handle a grounding concept gr(C) in the label of a node v with the tableau algorithm. Therefore, the gr-rule grounds the concept C based on the variable mappings that are associated to gr(C) on the node v.

Definition 9 (Grounding, Completion). For a concept C, Vars(C) is the set of nominal schema variables that syntactically occur in C. A concept C is grounded if |Vars(C)| = 0. Let µ be a variable mapping. We write C[µ] to denote the concept obtained by replacing each nominal schema {x} that occurs in C and x ∈ dom(µ) with the nominal [µ(x)].

Given a set of variables Y and a variable mapping set M with M* as the extension by the empty mapping ϵ if M = ∅, we define the completion comp_{k}^K(M) of M w.r.t. the variable set Y and a knowledge base K containing the individuals Inds(K) as

\[
\text{comp}_{k}^K(M) := \{\mu \cup \{x_1 \mapsto v_1, \ldots, x_n \mapsto v_n\} \mid \mu \in M^*, x_1, \ldots, x_n \in (Y \setminus \text{dom}(\mu)), v_1, \ldots, v_n \in \text{Inds}(K)\}.
\]

In order to ground the concept C for a concept fact gr(C)(v), the gr-rule uses the variable mappings of comp_{Vars(C)}^K(B(gr(C), v)). Since the mappings that are propagated
The set of variable mappings that is associated to a concept fact is shown in the super-role inclusion axioms [6]. Alternatively we could adapt the technique to eliminate role
\[ \forall v \{ \begin{array}{l}
\text{v is directly pairwise blocked by } w \text{ (see conditions 1 - 4 of Definition 6), and} \\
\mathcal{B}(C,v) = \mathcal{B}(C,w) \text{ and } \mathcal{B}(D,v') = \mathcal{B}(D,w') \text{ for all } C \in \mathcal{L}(v) \text{ and } D \in \mathcal{L}(v').
\end{array} \]
concepts in this example. However, since $O$ and thereby also the binder concepts are added to all ABox individuals, they automatically create variable mappings for every ABox individual. Obviously, many of these mappings are not necessary and their creation can easily be limited by additional axioms. For example, the variable $x$ only has to be bound if an $a$-neighbour and also a $v$-neighbour exists for an individual node, i.e., the binding of $x$ can be delayed with $\exists v. T \subseteq T'_1$, $\exists a. T \subseteq T'_2$, $(T'_1 \cap T'_2) \subseteq T'_3$ and $(O \cap T'_4) \subseteq \downarrow x. T$. The joins of the mapping sets are created in the nodes $a_1$ and $a_2$ for the concepts $T_3$ and $T_5$ and finally in node $a_0$ for the concept $T_7$. Only the variable mapping $\{x \mapsto a_1, y \mapsto a_3, z \mapsto a_4\}$ is propagated to the grounding concept $gr(\exists c.\{x\})$ and thus, by replacing the nominal schema $\{x\}$ with the nominal $\{a_1\}$, we have $\exists c.\{a_1\}$ as the only grounded concept. Hence, the individual $a_0$ is found to have a conflicting review assignment with the paper $a_1$.

4.3 Correctness

In the following we prove the correctness of our nominal schema absorption technique. For this, we roughly proceed as follows: Given a nominal schema axiom $C \subseteq D$ and an absorbed TBox $T$, then for $T_{ns}$ and $T_{ug}$ as the TBoxes obtained from absorbing $T \cup \{C \subseteq D\}$ and $T \cup \{U_1, \ldots, U_h\}$, respectively, where $U_1, \ldots, U_h$ are the upfront grounded axioms of $C \subseteq D$, we show that a fully expanded and clash free completion graph $G_{ns}$ for $T_{ns}$ can be converted to a fully expanded and clash free completion graph $G_{ug}$ for $T_{ug}$. Furthermore, we show that our extended tableau algorithm constructs a complete and clash free completion graph $G_{ns}$ for $T_{ns}$ if there exists a fully expanded and clash free completion graph $G_{ug}$ for $T_{ug}$ that is constructed by a standard tableau algorithm.

Please note that we only work with TBoxes instead of knowledge bases. This assumption is w.l.o.g. since in the presence of nominals ABoxes can be internalised (e.g., $C(a)$ is equivalent to the GCI $\{a \in A, r(a_1, a_2) \to \{a\} \in \exists r\{b\}$, etc.). We assume, therefore, that a completion $\text{comp}^{\text{fc}}(M)$ is analogously defined to the completion $\text{comp}^{\text{K}}(M)$ with $\mathcal{K} = (T, \emptyset)$.

To simplify the conversion between a completion graph for $T_{ns}$ and a standard completion graph for $T_{ug}$, we ensure that all concept facts can directly be converted into concept facts for the other completion graph. Therefore, we make the following simplifying assumptions: We assume that the absorption of nominals of the form $\neg\{a\}$ generates $\{a\} \subseteq \top \cap T$ instead of $\{a\} \subseteq T$ (cf. Algorithm 4, line 19), which is obviously logically equivalent. As a result, binder concepts such as $\downarrow x. T$ can be directly converted to concepts of the form $\top \cap T$. We also assume that the absorption of the upfront grounded axiom $C_{\{a\}} \subseteq D_{\{a\}}$, by the variable mapping $\mu$, creates a new special grounding concept $gr_\mu(D)$ to add the remaining, non-absorbable part of the axiom instead of directly implying $D_{\{a\}}$. This new concept construct retains the mapping $\mu$ and corresponds to the grounding concept $gr(D)$ that is created for the absorption of the nominal schema axiom $C \subseteq D$.

Before introducing the actual conversion, we first define the notion of concept and axiom set closure:

**Definition 11 (Closure).** The closure $\text{clos}(C')$ of a concept $C'$ is a set of concepts that is closed under sub-concepts of $C'$ and also contains $C'$. Additionally, $\text{tclos}(Z)$ is the
extension to a set of axioms \( Z \):

\[
\text{fclos}(Z) := \bigcup_{C \subseteq D \cap Z} \text{clos}(\neg C \sqcup D').
\]

For a TBox \( T \) and an axiom \( C' \subseteq D' \) with \( \text{nff}(\neg C') \) completely and \( D' \) not completely absorbable, the absorption closure \( \text{aclos}_T(C' \subseteq D') \) for \( T \) and \( C' \subseteq D' \) contains the new concepts introduced by the absorption of \( C' \subseteq D' \) and is defined as:

\[
\text{aclos}_T(C' \subseteq D') := \text{fclos}(X'_1, \ldots, X'_n) \setminus (\text{fclos}(T) \cup \text{clos}(D')),
\]

where \( X'_1, \ldots, X'_n \) are the axiom introduced by the absorption of \( C' \subseteq D' \).

Note that the concepts in the absorption closure are those that are relevant for the conversion between completion graphs since these are the concepts with variable mappings.

Now, the actual conversion of concepts and axioms obtained from the definition is as follows:

**Definition 12 (Conversion).** Let \( C \subseteq D \) be a nominal schema axiom where \( \text{nff}(\neg C) \) is completely and \( D \) not completely absorbable, and let \( \mu \) be a mapping with \( \text{dom}(\mu) = \text{Vars}(\neg C \sqcup D) \). Furthermore, let \( T \) be an absorbed TBox, \( T_\text{nff} \) and \( T_\text{ug} \) TBoxes obtained by absorbing \( T \cup \{C \subseteq D\} \) and \( T \cup \{U_1, \ldots, U_k\} \), respectively, where \( U_1, \ldots, U_k \) are the axioms obtained by the upfront grounding of \( C \subseteq D \). We denote the axioms (in creation order) and fresh atomic concepts obtained by absorbing \( \text{nff}(\neg C \cup D) \) with \( X_1, \ldots, X_n \) and \( T_1, \ldots, T_g \), respectively. Similarly, we use \( X'_1, \ldots, X'_n \) and \( T'_1, \ldots, T'_g \) for the case of absorbing \( \text{nff}(\neg C \cup D)\mu \).

For the concept \( C' \), we inductively define the concept conversion \( \text{conv}_\mu(C') \) of \( C' \) w.r.t. \( T, C \subseteq D \) and \( \mu \) as

\[
\text{conv}_\mu(C') := \begin{cases} C' & \text{if } C' \not\in \text{aclos}_T(C \subseteq D) \\ \left(\text{conv}_\mu(C'') \cap \text{clos}_\mu(D)\right) & \text{if } C' = \downarrow x.C'' \\ \text{gr}_\mu(D) & \text{if } C' = \text{gr}(D) \\ C'[T_1/T'_1, \ldots, T_g/T'_g] & \text{otherwise}, \end{cases}
\]

where \( C'[T_1/T'_1, \ldots, T_g/T'_g] \) denotes the syntactic replacement of each occurrence of \( T_i \) in \( C' \) with \( T'_i \), for \( 1 \leq i \leq g \). The extension to axioms \( \text{fconv}_\mu(X) \) is defined as:

\[
\text{fconv}_\mu(X) := \begin{cases} \{\mu(x)\} \subseteq \top \cap \text{conv}_\mu(D') & \text{if } X = O \subseteq \downarrow x.D' \\ \text{conv}_\mu(C') \subseteq \text{conv}_\mu(D') & \text{otherwise}. \end{cases}
\]

In the remainder of the section, we use \( C \subseteq D, \mu, T, T_\text{nff}, T_\text{ug}, T_1, \ldots, T_g, T'_1, \ldots, T'_g, X_1, \ldots, X_n, \) and \( X'_1, \ldots, X'_n \) as in the above definition.

Note that the restrictions on \( C \subseteq D \) are w.l.o.g. since any nominal schema axiom can be transformed into the desired form in an equivalence preserving manner. If \( \text{nff}(\neg C) \) is only partially absorbable, then a completely absorbable concept \( \text{nff}(\neg C') \) can be extracted from \( C \) (cf. Lemma 3), which can be used to obtain an axiom \( C' \subseteq D' \), where it holds that \( C' \subseteq C, C' \) is completely absorbable and \( D' = \text{nff}(\neg C') \cup D \) is
not completely absorbable. Also note that ¬T and ⊥ can always be used to extend a disjunction that corresponds to an axiom in order to obtain a completely absorbable and not completely absorbable disjunct w.r.t. our absorption algorithm.

We can now show that we can convert the axioms obtained by absorbing \( rnf(\neg C \sqcup D) \) from the nominal schema axiom \( C \subseteq D \) into the axioms that are obtained by absorbing the grounded version \( rnf((\neg C \sqcup D)_{[\mu]}) \), which is the first step in the conversion of a completion graph with nominal schema concepts to a standard completion graph:

**Lemma 4** Let \( \mathcal{T} \) be an absorbed TBox, \( C \subseteq D \) a nominal schema axiom, \( U_1, \ldots, U_h \) the upfront grounding, \( \mu \) a mapping, \( \mathcal{T}_{ns} \) and \( \mathcal{T}_{ns} \) TBoxes, and \( X_1, \ldots, X_n \) and \( X_1^\mu, \ldots, X_n^\mu \) axioms as in Definition 12. The set \( \{fconv_{\mu}(X_1), \ldots, fconv_{\mu}(X_n)\} \) is identical to the set \( \{X_1^\mu, \ldots, X_n^\mu\} \).

**Proof.** Let \( T_1, \ldots, T_n, T_1^\mu, \ldots, T_n^\mu \) be the fresh atomic concepts introduced by the absorption of \( rnf(\neg C \sqcup D) \) and \( rnf((\neg C \sqcup D)_{[\mu]}) \), respectively. Since the concepts \( rnf(\neg C \sqcup D) \) and \( rnf((\neg C \sqcup D)_{[\mu]}) \) only differ in the nominal schemas that are replaced by nominals, the absorption of \( rnf(\neg C \sqcup D) \) and \( rnf((\neg C \sqcup D)_{[\mu]}) \) is identical expect for axioms of the form \( Q \subseteq \bot \). Hence, by Definition 12, the claim holds. \( \square \)

For the conversion, we use the implicitly associated sets of variable mappings, which are defined as follows:

**Definition 13 (Implicitly Associated Mappings).** The implicitly associated set of variable mappings \( \text{mapp}^G(C'(v)) \) for a concept fact \( C'(v) \) and \( C' \) in the absorption closure w.r.t. a completion graph \( G = (V, E, \mathcal{L}, \hat{+}, \mathcal{B}) \) is defined as:

\[
\text{mapp}^G(C'(v)) := \begin{cases} 
\{x \mapsto v]\} & \text{if } C' = \bot \in D' \\
\mathcal{B}(C', v) & \text{if } \mathcal{B}(C', v) \neq \emptyset \\
\{v\} & \text{otherwise.}
\end{cases}
\]

Now, let \( G_{ns} \) be a completion graph showing the satisfiability of the TBox \( \mathcal{T}_{ns} \). We can replace each concept fact \( C'(v) \) with the implicitly associated variable mappings \( M \) and \( C' \in \text{aclos}_T(C \subseteq D) \), by the concept facts \( (\text{conv}_{\mu_1}(C'))(v), \ldots, (\text{conv}_{\mu_k}(C'))(v) \), where \( \mu_1, \ldots, \mu_k \) are the mappings obtained from the completion \( \text{comp}_{\text{Vars}(\neg C \sqcup D)}(M) \) of \( M \). As a result, we obtain a fully expanded completion graph \( G_{ag} \) that shows the satisfiability of the upfront grounded TBox \( \mathcal{T}_{ag} \).

**Lemma 5 (Soundness)** Let \( \mathcal{T} \) be an absorbed TBox, \( C \subseteq D \) a nominal schema axiom, \( U_1, \ldots, U_h \) the upfront grounding for \( C \subseteq D \), and \( \mathcal{T}_{ns} \) and \( \mathcal{T}_{ag} \) TBoxes as in Definition 12. If there is a fully expanded and clash free completion graph for \( \mathcal{T}_{ns} \), then there is a fully expanded and clash free completion graph for \( \mathcal{T}_{ag} \).

**Proof.** Let \( G_{ns} = (V_{ns}, E_{ns}, \mathcal{L}_{ns}, \hat{+}_{ns}, \mathcal{B}_{ns}) \) be a fully expanded and clash free completion graph for \( \mathcal{T}_{ns} \). We convert \( G_{ns} \) into a fully expanded and clash free completion graph \( G_{ag} \) by replacing every concept fact \( C'(v) \), \( C' \in \text{aclos}_T(C \subseteq D), v \in V_{ns} \), with the implicitly associated variable mappings \( M = \text{mapp}^{G_{ns}}(C'(v)) \), by the concept facts
the corresponding rules are not applicable for these concepts and axioms. Furthermore, let \(\varphi_1,\ldots,\varphi_l\) be all possible variable mappings for \(\text{Vars}(\sim C \cup D)\) w.r.t. \(T\), i.e., 
\[\{\varphi_1,\ldots,\varphi_l\} = \text{comp}_{\text{Vars}(\sim C \cup D)}^T(\epsilon)\].

In the following we show that none of the standard tableau rules for the concepts and axioms used in the absorption are applicable to \(G_{\text{ug}}\). Please note that the extended tableau rules (Table 2) coincide with the standard tableau rules (cf. Table 1) if no variable mappings are associated to the concept facts. Also note that the concept facts and axioms, which are not related to the absorption, are not affected by the conversion. Thus, the corresponding rules are not applicable for these concepts and axioms. Furthermore, if identical node labels are converted in the same way, blocking is not affected, i.e., if a node is blocked before the conversion, then it is also blocked after the conversion.

- We firstly consider the application of the \(\forall\)-rule, which is not applicable for \(G_{\text{ug}}\), because \(C' = \forall r \cdot D'(v)\) is converted to \((\text{conv}_{\lambda_1}(\forall r \cdot D'))(v), \ldots, (\text{conv}_{\lambda_1}(\forall r \cdot D'))(v)\) and for each \(r\)-neighbour node \(w\) of \(v\) the concept fact \(D'(w)\) is either also not associated with variable mappings (which is ensured by the absorption algorithm by creating separate axioms with fresh atomic concepts for the absorption of concepts that do not contain nominal schemas) or is at least also associated with the same variable mappings (otherwise the \(\forall\)-rule would be applicable for \(G_{\text{ug}}\) and thus \(D'(w)\) is at least also converted to \((\text{conv}_{\lambda_1}(D'))(w), \ldots, (\text{conv}_{\lambda_1}(D'))(w)\).

- We now consider the application of the \(\exists_1\)-rule. The absorption creates axioms of the form \(H \sqsubset D'\) with \(D' \in \text{aclos}_T(C \subseteq D)\) and \(H = \{a\}\) or \(H = A\). If \(D' \neq \downarrow x.D''\) (the replacement axioms for \(O \subseteq \downarrow x.D''\) are considered together with the \(\downarrow\)-concepts), \(H \notin \text{aclos}_T(C \subseteq D)\) and \(H = A\) or \(H = \{a\}\), then we would have the axioms \(H \in \text{conv}_{\lambda_1}(D'), \ldots, H \in \text{conv}_{\lambda_1}(D')\) in \(T_{\text{ug}}\) and the \(\exists_1\)-rule is not applicable, because, for every node \(v\) in \(G_{\text{ug}}\) with the concept fact \(H(v), D'(v)\) is also present and \(B_{\text{ug}}(D', v) = \emptyset\). Thus, \(D'(v)\) is replaced by \((\text{conv}_{\lambda_1}(D'))(v), \ldots, (\text{conv}_{\lambda_1}(D'))(v)\). If \(A \in \text{aclos}_T(C \subseteq D)\), then we would have the axioms \(\text{conv}_{\lambda_1}(A) \in \text{conv}_{\lambda_1}(D'), \ldots, \text{conv}_{\lambda_1}(A) \in \text{conv}_{\lambda_1}(D')\) and the \(\exists_1\)-rule is not applicable, because for every node \(v\) in \(G_{\text{ug}}\) with the concept fact \(A(v)\) and the associated variable mappings \(\mu_1, \ldots, \mu_k\), \(A(v)\) would be replaced by \((\text{conv}_{\mu_1}(A))(v), \ldots, (\text{conv}_{\mu_1}(A))(v)\), and \(D'\) is either also not associated with variable mappings (which is ensured by the absorption algorithm) or is at least also associated with the variable mappings \(\mu_1, \ldots, \mu_k\) (otherwise \(G_{\text{ug}}\) would not be fully expanded), and is at least also replaced by \((\text{conv}_{\mu_1}(D'))(v), \ldots, (\text{conv}_{\mu_1}(D'))(v)\). Thus, the \(\exists_1\)-rule is not applicable for \(G_{\text{ug}}\).

- Next, we consider the application of the \(\exists_2\)-rule for an axiom \((A_1 \cap A_2) \subseteq D'\). There are three cases:

1. If \(B_{\text{ug}}(A_1, v) = \emptyset\) and \(B_{\text{ug}}(A_2, v) = \emptyset\), then \(B_{\text{ug}}(D, v) = \emptyset\) and every concept fact \(D'(v)\) is replaced by \((\text{conv}_{\lambda_1}(D'))(v), \ldots, (\text{conv}_{\lambda_1}(D'))(v)\) and thus the rule is not applicable for \((A_1 \cap A_2) \subseteq \text{conv}_{\lambda_1}(D'), \ldots, (A_1 \cap A_2) \subseteq \text{conv}_{\lambda_1}(D')\).

2. If \(B_{\text{ug}}(A_1, v) \neq \emptyset\) (\(B_{\text{ug}}(A_2, v) \neq \emptyset\)), then the \(\exists_2\)-rule is analogously to the \(\exists_1\)-rule not applicable, because either there is no variable mapping that is associated to \(A_2(v)\) (\(A_1(v)\)) and, as a consequence, there is also no variable mapping associated to \(D'(v)\) (which is ensured by the absorption algorithm), or every variable mapping that is associated to \(A_2(v)\) (\(A_1(v)\)) is also associated to \(D'(v)\) if \(A_1(A_2)\) is also in the label of \(v\). Thus, the \(\exists_2\)-rule cannot add a \(\text{conv}_{\lambda_1}(D')\) concept to \(v\).
that is not already present, because the corresponding \( \text{conv}_i(A_1) \) (\( \text{conv}_i(A_2) \)) is missing.

3. If \( B_{ns}(A_1, v) \neq \emptyset \) and \( B_{ns}(A_2, v) \neq \emptyset \), the \( \Rightarrow_2 \)-rule is again not applicable after the conversion, because \( A_1(v) \) and \( A_2(v) \) are replaced by the concept facts (\( \text{conv}_i(A_1)(v) \)), \ldots, (\( \text{conv}_i(A_1))(v) \)) and (\( \text{conv}_i(A_2))(v) \), \ldots, (\( \text{conv}_i(A_2))(v) \), respectively, where \( \mu_1, \ldots, \mu_L \) and \( \mu'_1, \ldots, \mu'_L \) are the completion of the set of variable mappings \( \text{mapp}^{\text{gr}}(A_1(v)) \) and \( \text{mapp}^{\text{gr}}(A_2(v)) \). The \( \Rightarrow_2 \)-rule is, however, only applicable for an axiom (\( \text{conv}_i(A_1) \cap \text{conv}_i(A_2) \)) \( \subseteq \text{conv}_i(D') \) if \( \text{conv}_i(A_1) \) as well as \( \text{conv}_i(A_2) \) is in the same label, but \( \text{conv}_i(D') \) is not already present, i.e., \( \mu \in \{ \mu_1, \ldots, \mu_L \} \) and \( \mu \in \{ \mu'_1, \ldots, \mu'_L \} \), but \( \mu \notin \{ \mu_1, \ldots, \mu_L \} \) \( \Rightarrow_2 \) \( \{ \mu'_1, \ldots, \mu'_L \} \), which is a contradiction, because \( \{ \mu_1, \ldots, \mu_L \} \) \( \Rightarrow_2 \) \( \{ \mu'_1, \ldots, \mu'_L \} \) is the same as the completion of \( B_{ns}(A_1, v) \) \( \Rightarrow_2 \) \( B_{ns}(A_2, v) \) to all possible variables used in \( C \subseteq D \).

- The \( \downarrow \)-concepts are more complicated. Concept facts of the form \( \downarrow x.D'(a) \) are not explicitly associated with variable mappings. However, because of the axiom \( O \subseteq \downarrow x.D' \), they only occur in the label of ABox individual nodes. Thus, we can use the implicit information that \( x \) will be bound to the ABox individual node \( a \), and we use the completion of the variable mapping \( \{ x \mapsto a \} \) for \( \mu_1, \ldots, \mu_L \). Therefore, we replace \( \downarrow x.D'(a) \) with the concept facts (\( \top \sqcap \text{conv}_i(D')(a) \)), \ldots, (\( \top \sqcap \text{conv}_i(D')(a) \)). It is not hard to see that \( (\top \sqcap \text{conv}_i(D')) \), \ldots, (\( \top \sqcap \text{conv}_i(D') \)) cannot be unfolded in \( G_{ag} \), because the \( \downarrow \)-rule ensures that \( D' \) is already present in the label of the node and is associated with the variable mapping \( \{ x \mapsto a \} \) and, thus, \( D' \) is also replaced by \( \text{conv}_i(D') \), \ldots, \( \text{conv}_i(D') \). Analogously, for the axioms \( [a] \subseteq \top \sqcap \text{conv}_i(D') \), \ldots, \( [a] \subseteq \top \sqcap \text{conv}_i(D') \) that we have to consider in \( G_{ag} \) instead of \( O \subseteq \downarrow x.D' \), the rules for these axioms are also not applicable, because the concept \( \downarrow x.D' \) in the label of \( a \) has been replaced by the concepts \( \top \sqcap \text{conv}_i(D') \), \ldots, \( \top \sqcap \text{conv}_i(D') \) and \( \downarrow x.D' \) is in the label of \( a \), because it is added to every ABox individual node due to the axiom \( O \subseteq \downarrow x.D' \).

- The argumentation for the \( \text{gr} \)-concepts and the corresponding rules is very similar. As mentioned before, we assume that the grounding concept is always used to add the remaining, non-absorbable part of the axiom. Thus, \( \text{gr}(D) \) is always in \( \text{acos}_T(C \subseteq D) \), even if \( \text{Vars}(D) = \emptyset \). Furthermore, we also use the assumption that the absorption of an upfront grounded axiom, by the variable mapping \( \mu \), also uses a special grounding concept \( \text{gr}_\mu(D) \), which has to be unfolded to \( D_{[\mu]} \) and is, therefore, not problematic for the tableau algorithm, because it corresponds to a conjunction with only one conjunct. Thus, a concept fact \( \text{gr}(D)(v) \) is replaced by (\( \text{conv}_i(\text{gr}(D))(v) \)), \ldots, (\( \text{conv}_i(\text{gr}(D))(v) \)), which is the same as (\( \text{gr}_\mu(D))(v) \), \ldots, (\( \text{gr}_\mu(D))(v) \)). Obviously, these replaced grounding concepts cannot be unfolded to \( D_{[\mu]}, \ldots, D_{[\mu]} \), because \( D_{[\mu]}, \ldots, D_{[\mu]} \) are already present due to the application of the \( \text{gr} \)-rule for \( \text{gr}(D)(v) \), for which also the completion of the associated set of variable mappings is used for the grounding of \( D \).

Next, we show that we can steer our extended tableau algorithm to construct a complete and clash free completion graph \( G_{ag} \) for \( T_{ns} \) if there exists a fully expanded and clash free completion graph \( G_{ag} \) for \( T_{ag} \) that is constructed by a standard tableau algorithm.

28
Lemma 6 (Completeness) Let $\mathcal{T}$ be an absorbed TBox, $C \subseteq D$ a nominal schema axiom, $U_1, \ldots, U_k$ the up-front grounding for $C \subseteq D$, and $\mathcal{T}_{na}$ and $\mathcal{T}_{ag}$ TBoxes as in Definition 12. If there is a fully expanded and clash-free completion graph for $\mathcal{T}_{ag}$, then there is a fully expanded and clash-free completion graph for $\mathcal{T}_{na}$.

Proof. Let $G_{ag}$ be a completion graph for $\mathcal{T}_{ag}$ that is obtained by applying only rules for concepts and axioms of $\mathcal{T}$. Since our extended rules coincide with the standard tableau rules if no variable mappings are associated to concept facts, our extended tableau algorithm can create $G_{na}$, which exactly coincides with $G_{ag}$. We show that the application of a rule in Table 2 to $G_{na}$ deterministically adds only concept facts and possibly variable mappings, for which the conversion of these facts and variable mappings are also consequences in $G_{ag}$ that are added in the course of applying standard tableau rules to $G_{ag}$. Thus, $G_{ag}$ can obviously be used for steering the non-deterministic decisions for $G_{na}$ to construct a fully expanded and clash-free completion graph if $G_{ag}$ is fully expanded and clash free.

Now, let $G_{na}$ and $G_{ag}$ be completion graphs for $\mathcal{T}_{na}$ and $\mathcal{T}_{ag}$, respectively, and $G_{na}$ and $G_{ag}$ coincide with the inferred facts so far, i.e., the conversion of concept facts and variable mappings from $G_{na}$ corresponds to the contained concept facts in $G_{ag}$. To show by induction that each rule application for $G_{na}$ only adds concept facts and variable mappings, for which the conversion of these facts and variable mappings are also consequences in $G_{ag}$, let $\lambda_1, \ldots, \lambda_\ell$ be all possible variable mappings, i.e., $\{\lambda_1, \ldots, \lambda_\ell\} = \text{comp}^T_{\text{Vars}(C \cup D)}(\{\epsilon\})$. Please note, it suffices to consider only the extended rules for concepts and axioms used for absorbing $C \subseteq D$, because only the concepts in $\text{aclos}_{\mathcal{T}}(C \subseteq D)$ can be associated with variable mappings, for which the extended rules differ to standard rules.

- First, we consider the $\forall$-rule for a concept fact $\forall r.D'(v)$, $\forall r.D' \in \text{aclos}_{\mathcal{T}}(C \subseteq D)$, which adds the concept fact $D'(w)$ to an $r$-neighbour $w$ of $v$ in $G_{na}$ and possibly the variable mapping $\mu \in B_{na}(\forall r.D', v)$ to $B_{ag}(D', w)$. If the $\forall$-rule only adds the concept fact $D'(w)$ for cases where $B(\forall r.D', v) = \emptyset$, then $\text{mapp}^{\downarrow \text{na}}(D'(w)) = \{\epsilon\}$ (which is ensured by the absorption algorithm) and we have to show that in the completion graph $G_{ag}$ the concept facts $(\text{conv}_{\lambda_\ell}(D'))(w), \ldots, (\text{conv}_{\lambda_\ell}(D'))(w)$ are also added by rule applications. Obviously, this is the case, because the concept fact $\forall r.D'(v)$ corresponds to $(\text{conv}_{\lambda_\ell}(\forall r.D'))(v), \ldots, (\text{conv}_{\lambda_\ell}(\forall r.D'))(v)$ in $G_{ag}$ and by applying the $\forall$-rule for all concept facts $(\text{conv}_{\lambda_\ell}(\forall r.D'))(v), 1 \leq j \leq \ell$, we have the concepts $\text{conv}_{\lambda_j}(D'), \ldots, \text{conv}_{\lambda_\ell}(D')$ in the label of all neighbour nodes. If the $\forall$-rule adds a variable mapping $\mu \in B_{ag}(\forall r.D', v)$ to $B_{ag}(D', w)$, then we have to show that $(\text{conv}_{\lambda_\ell}(D'))(w), \ldots, (\text{conv}_{\lambda_\ell}(D'))(w)$ with $[\mu_1, \ldots, \mu_k] = \text{comp}^T_{\text{Vars}(C \cup D)}(\{\mu\})$ are added to $G_{ag}$ by rule applications. But this is also the case since $\forall r.D'(v)$ corresponds to $(\text{conv}_{\lambda_\ell}(\forall r.D'))(w), \ldots, (\text{conv}_{\lambda_\ell}(\forall r.D'))(w)$ in $G_{ag}$ and applying the $\forall$-rule for $(\text{conv}_{\lambda_\ell}(\forall r.D'))(w), \ldots, (\text{conv}_{\lambda_\ell}(\forall r.D'))(w)$ adds $(\text{conv}_{\mu_1}(D'))(w), \ldots, (\text{conv}_{\mu_\ell}(D'))(w)$ to the label of all neighbour nodes.
- Next, we consider the $\sqsubseteq_1$-rule for an axiom $H \sqsubseteq D'$ with $H = A$ or $H = \{a\}$ and $D' \neq \downarrow_1 D''$ (we consider the addition of the binder concepts together with the $\downarrow$-rule). If $B_{na}(H, v) = \emptyset$ and the $\sqsubseteq_1$-rule adds only the concept fact $D'(v)$ to a node, then we have to show that $(\text{conv}_{\lambda_\ell}(D'))(v), \ldots, (\text{conv}_{\lambda_\ell}(D'))(v)$ are also
Let us now consider the $\Sigma_2$-rule for an axiom $(A_1 \land A_2) \subseteq D'$. If the $\Sigma_2$-rule only adds the concept fact $D'(v)$, then we have to show that $(\text{conv}_{\mu_1}(D'(v)), \ldots, \text{conv}_{\mu_\ell}(D'(v)))$ are also added to $G_{\text{ug}}$ by rule applications. However, this is the case, because the $\Sigma_2$-rule adds also the variable mapping $\mu$ to $B_{\text{nt}}(D', v)$, and since we have the axiom $(\text{conv}_{\mu_1}(A_1) \cap \text{conv}_{\mu_2}(A_2)) \subseteq \text{conv}_{\mu_1}(D)$ for each $1 \leq j \leq \ell$, it follows that all $(\text{conv}_{\mu_1}(D'(v)), \ldots, \text{conv}_{\mu_\ell}(D'(v)))$ are also added to $G_{\text{ug}}$. If the $\Sigma_2$-rule also adds the variable mapping $\mu$ to $B_{\text{nt}}(D', v)$, then we have to show that $(\text{conv}_{\mu_1}(D'(v)), \ldots, \text{conv}_{\mu_\ell}(D'(v)))$ with $\{\mu_1, \ldots, \mu_\ell\} = \text{comp}_{\text{Var}^\text{FOL} - \text{U}
abla \text{U}
abla}(\{\mu\})$ are also added to $G_{\text{ug}}$ by rule applications. Let us first assume that $B_{\text{nt}}(A_1, v) = \emptyset$ ($B_{\text{nt}}(A_2, v) = \emptyset$). As a consequence, we have in $G_{\text{ug}}$ the concept facts $(\text{conv}_{\mu_1}(A_1)), \ldots, \text{conv}_{\mu_\ell}(A_1)) \subseteq \text{conv}_{\mu_1}(D)$, and as a consequence of the axioms $(\text{conv}_{\mu_1}(A_1) \cap \text{conv}_{\mu_2}(A_2)) \subseteq \text{conv}_{\mu_1}(D)$, for all $1 \leq j \leq \ell$, the concept facts $(\text{conv}_{\mu_1}(D'(v)), \ldots, \text{conv}_{\mu_\ell}(D'(v)))$ are also added to $G_{\text{ug}}$ by rule applications. Let us now assume that $B_{\text{nt}}(A_1, v) \neq \emptyset$ as well as $B_{\text{nt}}(A_2, v) \neq \emptyset$. We show that $(\text{conv}_{\mu_1}(D'(v)), \ldots, \text{conv}_{\mu_\ell}(D'(v)))$ has to be added to $G_{\text{ug}}$, because $(\text{conv}_{\mu_1}(A_1)), \ldots, \text{conv}_{\mu_\ell}(A_1))$ as well as $(\text{conv}_{\mu_1}(A_2)), \ldots, \text{conv}_{\mu_\ell}(A_2))$ are in $G_{\text{ug}}$. Obviously, there exists the variable mappings $\mu' \in B_{\text{nt}}(A_1, v)$ and $\mu'' \in B_{\text{nt}}(A_2, v)$ with $\mu = \text{dom}(\mu') \cup \text{dom}(\mu'')$ and for each $x \in \text{dom}(\mu') \cap \text{dom}(\mu'')$ it holds that $\mu'(x) = \mu''(x)$. Thus, $\mu' \subseteq \mu$ and $\mu'' \subseteq \mu$ and as a consequence of the completion of $\mu'$ and $\mu''$ it follows that $\{\mu_1, \ldots, \mu_\ell\} \subseteq \{\mu'_1, \ldots, \mu'_\ell\}$ and $\{\mu'_1, \ldots, \mu'_\ell\} \subseteq \{\mu''_1, \ldots, \mu''_\ell\}$. Therefore, $(\text{conv}_{\mu_1}(A_1)) \cup \ldots \cup (\text{conv}_{\mu_\ell}(A_2)) \subseteq (\text{conv}_{\mu_1}(A_1)) \cup \ldots \cup (\text{conv}_{\mu_\ell}(A_2))$ of at least in $G_{\text{ug}}$

The $\lambda$-rule for a concept fact $\lambda x.D'(a)$ adds $D'$ to the label of $a$ and the variable mapping $\{x \mapsto a\}$ to $B_{\text{nt}}(D', a)$. We have to show that the concept facts $(\text{conv}_{\mu_1}(D'(a)), \ldots, \text{conv}_{\mu_\ell}(D'(a)))$ with $\{\mu_1, \ldots, \mu_\ell\} = \text{comp}_{\text{Var}^\text{FOL} - \text{U}
abla \text{U}
abla}([x \mapsto a])$ are also added to $G_{\text{ug}}$ by rule applications. But this is obvious, because in $G_{\text{ug}}$ we have the concept facts $(\text{conv}_{\mu_1}(\lambda x.D'(a)), \ldots, \text{conv}_{\mu_\ell}(\lambda x.D'(a)))$, which is nothing else than $(\top \cap \text{conv}_{\mu_1}(D'(a)), \ldots, \top \cap \text{conv}_{\mu_\ell}(D'(a)))$. Furthermore, we have to show that $(\top \cap \text{conv}_{\mu_1}(D'(a)), \ldots, \top \cap \text{conv}_{\mu_\ell}(D'(a)))$ is added to $G_{\text{ug}}$, because, as a consequence of the axiom $O \subseteq \lambda x.D', \lambda x.D'(a)$ is added to $G_{\text{ug}}$. Obviously, this is the case, because for $G_{\text{ug}}$ we have the axioms $[a] \subseteq \lambda x.(\text{conv}_{\mu_1}(D'), \ldots, [a] \subseteq \lambda x.(\text{conv}_{\mu_1}(D'),$

The application of the $gr$-rule adds for a concept fact $gr(D'(v))$ and a (possibly empty) variable mapping $\mu \in B_{\text{nt}}(gr(D'(v)), v)$ the concept facts $D_{\mu_1}, \ldots, D_{\mu_\ell}$ with $\{\mu_1, \ldots, \mu_\ell\} = \text{comp}_{\text{Var}^\text{FOL} - \text{U}
abla \text{U}
abla}(\{\mu\})$. We have to show that $D_{\mu_1}, \ldots, D_{\mu_\ell}$ is also added to $G_{\text{ug}}$ by rule applications. Again, this is obviously the case, because in
we have the concept facts \(\text{conv}_{\mu_1}(\text{gr}(D'))(v), \ldots, \text{conv}_{\mu_k}(\text{gr}(D'))(v)\), which is the same as \(\text{gr}_{\mu_1}(D')(v), \ldots, \text{gr}_{\mu_k}(D')(v)\). □

The extended tableau algorithm is still terminating. This is due to the fact that the number of variable mappings is limited by the number of ABox individuals and the number of variables in axioms. Thus, blocking is ensured since the nodes in the completion graph can only be labelled with a limited number of concepts and only a limited number of variable mappings can be associated to these concepts.

**Lemma 7 (Termination)** Let \(\mathcal{L}\) be a Description Logic without nominal schemas and \(\mathcal{L}'\mathcal{V}\) its extension with nominal schemas. Extending a tableau decision procedure for the satisfiability of \(\mathcal{L}\)-TBoxes based on the rules of Table 1 with the rules of Table 2 results in a terminating algorithm for absorbed \(\mathcal{L}'\mathcal{V}\)-TBoxes.

As a result, we obtain a terminating tableau algorithm that is sound and complete for absorbed TBoxes with nominal schema axioms:

**Theorem 2** Let \(\mathcal{L}\) be a Description Logic without nominal schemas and \(\mathcal{L}'\mathcal{V}\) its extension with nominal schemas. Extending a tableau decision procedure for satisfiability of \(\mathcal{L}\)-TBoxes based on the rules of Table 1 with the rules of Table 2 yields a decision procedure for the satisfiability of absorbed \(\mathcal{L}'\mathcal{V}\)-TBoxes.

5 Backward Chaining Optimisations

In comparison to the upfront grounding approach, the nominal schema absorption is usually a huge improvement for knowledge bases, where axioms with absorbable nominal schemas do not match to every combination of ABox individuals. However, the propagation of variable mappings can still lead to practical problems. On the one hand, it is unfavourable that the mappings are created and propagated to many nodes and even to such nodes, where the conditions of the absorptions cannot be satisfied. On the other hand, if there are several neighbour nodes that satisfy some absorption condition, then the join potentially creates quite a lot of new mappings.

**Example 4.** In order to illustrate the problems, let us assume that we have an axiom 
\[
\exists r.\{x\} \sqcap \exists s.\{y\} \sqcap A \sqsubseteq B,
\]
which is absorbed as follows:
\[
\begin{align*}
O &\sqsubseteq \downarrow r.T_x & T_x &\sqsubseteq \forall r.\downarrow T_1 & O &\sqsubseteq \downarrow y.T_y \\
T_x &\sqsubseteq \forall s.\downarrow T_2 & (T_1 \sqcap T_2) &\sqsubseteq T_3 & (T_3 \sqcap A) &\sqsubseteq T_4 \\
T_4 &\sqsubseteq \text{gr}(B).
\end{align*}
\]
Moreover, the knowledge base contains the assertions:
\[
\begin{align*}
 r(c_0, a_0) & & r(c_0, a_1) & & s(c_0, b_0) & & s(c_0, b_1).
\end{align*}
\]
Subsequently, the completion graph in Figure 2 is generated by testing the consistency of the knowledge base. Obviously, both of the aforementioned problems occur in the generated completion graph. Due to the missing concept \(A\) in the label of node \(c_0\), it is impossible to propagate the variable mappings to the grounding concept \(\text{gr}(B)\). Nevertheless, the algorithm creates mappings with new bindings for the variables \(x\) and \(y\) for each node. Furthermore, the concept \(T_3\) in the label of \(c_0\) is already associated
variable mappings such that only those mappings are combined, which represent the assertions never­theless possible to optimise the creation and combination of variable mappings. Before the grounding concept can be implied, if we can use the assertions \( r(c_1), a_1, s(c_1, b_1) \) and \( A(c_1) \) (cf. Figure 2, extension in dashed lines). Now, the basic idea is to first detect, with a simpler method, “interesting” nodes that can satisfy the conditions of the absorption, i.e., nodes that can possibly have the grounding concept in their label, and also those ABox individuals that might be “candidates” for binding the nominal schema variables.

Let us assume that we are able to detect \( c_1 \) as “interesting” with \( a_1 \) as a “candidate” for \( x \) and \( b_1 \) as a “candidate” for \( y \). We now propagate these binding candidates back from the interesting node to the concepts and nodes that are relevant to imply the grounding concept for \( c_1 \). For example, the axiom \( T_1 \cap T_2 \subseteq T_3 \) tells us that the variable mappings that are associated to \( T_1 \) and \( T_2 \) on the node \( c_1 \) have to be joined and then propagated to \( T_3 \) before the grounding concept can be implied. If we can use the information of the back propagated binding candidates, then we can limit the join of variable mappings such that only those mappings are combined, which represent the expectation of the candidates, i.e., we combine only the mapping \( \mu_1 \) and \( \mu_2 \) for which we know that for every \( z \in \text{dom}(\mu_1 \cup \mu_2), (\mu_1 \cup \mu_2)(z) \) is a back propagated candidate for \( z \). Thus, we can control the join with the back propagated binding candidates. Of course, to retain completeness, the set of binding candidates must be a superset of the those bindings in variable mappings that are indeed required to ground all necessary concepts. Moreover, we can further propagate the candidates back over the \( r \) and \( s \) roles to control the binder concepts, which is, as a consequence of the axioms \( T_r \subseteq \forall r^{-}.T_1 \) and \( T_s \subseteq \forall s^{-}.T_2 \), also a requirement to imply the grounding concept.

Fig. 2. Naive propagation and resulting combinatorial explosion of variable mappings

with four new variable mappings that are created by joining the mappings associated to \( T_1 \) and \( T_2 \).

Although the problems cannot be completely avoided in worst-case scenarios, it is nevertheless possible to optimise the creation and combination of variable mappings with backward chaining for many practical knowledge bases. In Example 4, no individual can ever have the grounding concept in its label. To make the example more interesting, let us assume that the knowledge base of Example 4 is extended by the assertions \( r(c_1, a_1), s(c_1, b_1) \) and \( A(c_1) \) (cf. Figure 2, extension in dashed lines). Now, the basic idea is to first detect, with a simpler method, “interesting” nodes that can satisfy the conditions of the absorption, i.e., nodes that can possibly have the grounding concept in their label, and also those ABox individuals that might be “candidates” for binding the nominal schema variables.

Let us assume that we are able to detect \( c_1 \) as “interesting” with \( a_1 \) as a “candidate” for \( x \) and \( b_1 \) as a “candidate” for \( y \). We now propagate these binding candidates back from the interesting node to the concepts and nodes that are relevant to imply the grounding concept for \( c_1 \). For example, the axiom \( T_1 \cap T_2 \subseteq T_3 \) tells us that the variable mappings that are associated to \( T_1 \) and \( T_2 \) on the node \( c_1 \) have to be joined and then propagated to \( T_3 \) before the grounding concept can be implied. If we can use the information of the back propagated binding candidates, then we can limit the join of variable mappings such that only those mappings are combined, which represent the expectation of the candidates, i.e., we combine only the mapping \( \mu_1 \) and \( \mu_2 \) for which we know that for every \( z \in \text{dom}(\mu_1 \cup \mu_2), (\mu_1 \cup \mu_2)(z) \) is a back propagated candidate for \( z \). Thus, we can control the join with the back propagated binding candidates. Of course, to retain completeness, the set of binding candidates must be a superset of the those bindings in variable mappings that are indeed required to ground all necessary concepts. Moreover, we can further propagate the candidates back over the \( r \) and \( s \) roles to control the binder concepts, which is, as a consequence of the axioms \( T_r \subseteq \forall r^{-}.T_1 \) and \( T_s \subseteq \forall s^{-}.T_2 \), also a requirement to imply the grounding concept.
We next describe a more systematic approach to this idea of back propagation from nodes such as $c_1$ in the above example. A nice feature of this approach is that our absorption algorithms only require slight modifications for this purpose.

In the following, we extend the tableau and absorption algorithm such that we can propagate candidates for variables and use the (back) propagated candidates to control the creation and combination of variable mappings. For now, let us assume that for each nominal schema variable $x$ all individuals $a_1, \ldots, a_m$ which are a candidate for $x$, are already identified. Thus, we get the binding candidates $x/a_1, \ldots, x/a_m$ and we encode these bindings also as variable mappings, i.e., if $a_j$ is an individual that is a candidate for the variable $x$, then we encode this in the variable mapping $\{x \mapsto a_j\}$. Hence, we can also use the rules in Table 2 for the back propagation of the binding candidates. To consider the back propagated bindings in the $\perp$- and $\sqsubseteq$-rules, it is necessary to know to which $\perp x. C$ concepts and $(A_1 \sqcap A_2) \sqsubseteq C$ axioms the back propagated bindings are associated. Moreover, to distinguish the previously introduced concepts (rules) from the new ones for the optimisation, we use $\perp_{BP} x. C$ and $(A_1, A_2) \rightarrow_{BP} C$ for the new binder and join concepts ($\perp_{BP}$ and $\rightarrow_{BP}$ for the new rules), where $BP$ denotes the considered back propagation. Note, instead of using a binary absorption axiom of the form $(A_1 \sqcap A_2) \rightarrow_{BP} C$, we now use the concept $(A_1, A_2) \rightarrow_{BP} C$, because then the binding candidates can be directly propagated to the concept $(A_1, A_2) \rightarrow_{BP} C$ and this makes it easier to consider the candidates in the associated rule.

The new rules with the considered backward chaining propagation are shown in Table 3. The main difference is that now only those variable mappings are created and combined, for which all the individuals can be found in the back propagated binding candidates. Thus, the $\perp_{BP}$-rule creates a new variable mapping $\{x \mapsto v\}$ on a node $v$ only if the concept $\perp_{BP} x. C$ is in the label of $v$ and the backward chaining has propagated the binding candidate $x/v$ to the concept $\perp_{BP} x. C$ in the node $v$. Since the binding $x/v$ is also propagated as mapping, we can use $\{x \mapsto v\} \in B(\perp_{BP} x. C, v)$ as condition to trigger the application of the $\perp_{BP}$-rule. Analogously, the $\rightarrow_{BP}$-rule only creates the join of the variable mappings $\mu_1$ and $\mu_2$ on a node $v$ if all individuals for the variables can be found in the back propagated bindings, i.e., $\{x \mapsto (\mu_1 \cup \mu_2)(x)\} \in B((A_1, A_2) \rightarrow_{BP} C, v)$ for every $x \in dom(\mu_1 \cup \mu_2)$.

In order to support the backward chaining propagation in the absorption algorithm, some minor adjustments in the absorbJoined and absorbConcept functions are nec-

| $\rightarrow_{BP}$-rule: if | $(A_1, A_2) \rightarrow_{BP} C \in L(v), [A_1, A_2] \subseteq L(v), v$ not indirectly blocked, and |
| if | $\mathcal{B}(A_1, v) \cup \mathcal{B}(A_2, v) = \emptyset, C \notin L(v)$ |
| then | $L(v) \rightarrow L(v) \cup \{C\}$ |
| if | there exist $\mu \in (\mathcal{B}(A_1, v) \triangleright \mathcal{B}(A_2, v))$ with $\{x \mapsto \mu(x)\} \in \mathcal{B}((A_1, A_2) \rightarrow_{BP} C, v)$ for all $x \in \text{dom}(\mu)$, $C \notin L(v)$ or $\mu \notin \mathcal{B}(C, v)$ |
| then | $L(v) \rightarrow L(v) \cup \{C\}$ and $\mathcal{B}(C(v)) \rightarrow \mathcal{B}(C(v)) \cup \{\mu\}$ |

| $\perp_{BP}$-rule: if | $\perp_{BP} x. C \in L(v), v$ not indirectly blocked, $\{x \mapsto v\} \in \mathcal{B}(\perp_{BP} x. C, v)$, and $C \notin L(v)$ or $\{x \mapsto v\} \notin \mathcal{B}(C, v)$ |
| then | $L(v) \rightarrow L(v) \cup \{C\}$ and $\mathcal{B}(C, v) \rightarrow \{\{x \mapsto v\}\}$ |
The algorithm is called with absorbJoinedBP((∀₁₂, ∀₁₂), ∅) and in the new absorbConceptBP function. Both new functions are extended to create the concepts and axioms for the back propagation of binding candidates during the absorption.

Again, absorbJoinedBP is joining several atomic concepts, which are created by absorbConcept or absorbConceptBP. Additionally, absorbJoinedBP (Algorithm 8) distributes the bindings candidates, which were already propagated back to the concept A_{BP}, to all join concepts of the form (A_1, A_2) →_{BP} C (line 13). Since not all concepts of S necessarily still contain nominal schema variables, the concepts without nominal schema variables are absorbed as before by the absorbJoined and absorbConcept functions (lines 2-6).

The adjustment of absorbConceptBP in Algorithm 9 is very similar. The function is also called with an additional atomic concept A_{BP}, for which the back propagation for the new absorption is appended. As an example, for the absorption of a ∀r.C concept, the back propagation is extended with the propagation over an r-edge to a fresh atomic concept T_{BP-nb} (lines 2-3). The bindings that are back propagated to T_{BP-nb} can then be used to control the join or also to limit the creation of new variable mappings.

Example 5. If we absorb the axiom ∃r.[x] ∩ ∃s.[y] ⊑ A ⊑ B of Example 4 with the new absorption algorithms, which also generate the back propagation of binding candidates, then we obtain the following axioms:

| T_{BP1} | ∀r.T_{BP2} | (O ⊓ T_{BP2}) ⊑ [x],T_x | T_x ⊑ ∀r'.T_1 |
| T_{BP1} | ∀y.T_{BP3} | (O ⊓ T_{BP3}) ⊑ [y],T_y | T_y ⊑ ∀s'.T_2 |
| T_{BP1} | (T_1, T_2) →_{BP} T_3 | T_{BP1} ⊑ (T_3, A) →_{BP} T_4 | T_4 ⊑ gr(B) |

The algorithm is called with T_{BP1} as the initial atomic concept for the back propagation, i.e., absorbJoinedBP((∀₁₂, ∀₁₂), ∀₁₂, ∅), T_{BP1}), and T_{BP2}, T_{BP3} are atomic-con-
Algorithm 9 absorbConcept\textsubscript{BP}(C, A\textsubscript{BP})

Output: Returns the atomic concept for the absorption of C

1: if \( C = \forall r.C' \) then
2: \( T\textsubscript{BP} - nb \leftarrow \) fresh atomic concept
3: \( T' \leftarrow T' \cup \{ A\textsubscript{BP} \sqsubseteq \exists r.T\textsubscript{BP} - nb \} \)
4: \( A\textsubscript{nb} \leftarrow \) absorbJoined\textsubscript{BP}(\text{collectDisjuncts}(C', \text{true}), T\textsubscript{BP} - nb)
5: \( T \leftarrow \) fresh atomic concept
6: \( T' \leftarrow T' \cup \{ A\textsubscript{nb} \sqsubseteq \exists r.T \} \)
7: return \( T \)
8: else if \( C = \leq n r.C' \) then
9: \( T\textsubscript{BP} - nb \leftarrow \) fresh atomic concept
10: \( T' \leftarrow T' \cup \{ A\textsubscript{BP} \sqsubseteq \exists r.T\textsubscript{BP} - nb \} \)
11: \( A\textsubscript{nb} \leftarrow \) absorbJoined\textsubscript{BP}(\text{collectDisjuncts}(\text{nnf}(\neg C'), \text{true}), T\textsubscript{BP} - nb)
12: \( T \leftarrow \) fresh atomic concept
13: \( T' \leftarrow T' \cup \{ A\textsubscript{nb} \sqsubseteq \exists r.T \} \)
14: return \( T \)
15: else if \( C = \neg \{ \exists x \} \) then
16: \( T\textsubscript{x} \leftarrow \) fresh atomic concept
17: \( T' \leftarrow T' \cup \{ O \sqcap A\textsubscript{BP} \sqsubseteq \downarrow\textsubscript{BP} x.T\textsubscript{x} \} \)
18: return \( T\textsubscript{x} \)
19: end if

cepts that are additionally created for the back propagation of binding candidates. Now, if we ensure that such binding candidates are automatically propagated to \( T\textsubscript{BP} \), then only the desired mappings are propagated to the grounding concept and the creation, combination and propagation of other mappings can be limited significantly.

5.1 Identification of Interesting Nodes and Binding Candidates

So far, we have assumed that the interesting nodes and the binding candidates used for the backward chaining are already available. In the following, we present different approaches of how these interesting nodes and bindings candidates can be identified.

A very simple, but already very effective method is to first absorb the absorbable part of an axiom \( W \), where all nominal schemas are replaced by \( O \). In comparison to the absorption of the original nominal schema axiom, we do not generate a grounding concept since we are only interested in the atomic concept, say \( A\textsubscript{O} \), which is generated for absorbing \( W \), where the nominal schemas are replaced by \( O \). During the expansion of a completion graph, the concept \( A\textsubscript{O} \) now marks all nodes for which it is possible that the variable mappings are propagated to the grounding concept \( gr(C) \) that is created for the absorption of the nominal schema axiom \( W \). Obviously, if \( A\textsubscript{O} \) is not in the label of a node \( v \), then there is no variable mapping, which could be propagated to \( gr(C) \), because the absorption with \( O \) is more general by allowing every combination of ABox individuals to match the conditions of the absorption. Thus, the concept \( A\textsubscript{O} \) marks interesting nodes, which are good candidates to start the backward chaining propagation.
As a result, only those variable mappings of $B(A_1, v)$ and $B(A_2, v)$ are filtered, which are using variables that occur in the mappings of $B(A_1, v)$ as well as in the mappings of $B(A_2, v)$. Note, the $\rightarrow_{cp}$-rule does not combine or create new variable mappings.
and, thus, the mappings that are propagated to \( B(C, v) \) also map only one variable to an individual name if the mappings of \( B(A_1, v) \) and \( B(A_2, v) \) map only one variable to an individual name. Thus, the mappings that are created by the binder concepts can be filtered with the \( \to_{\text{BP}} \) rule and we create the desired binding candidates encoded as mappings that map one variable to an individual name. For example, if the mappings \( \{ x \mapsto a_1 \}, \{ x \mapsto a_2 \}, \{ y \mapsto b \} \) are associated to \( A_1 \) and \( \{ x \mapsto a_1 \}, \{ z \mapsto c \} \) are associated to \( A_2 \), then the \( \to_{\text{BP}} \) rule propagates \( \{ x \mapsto a_1 \}, \{ y \mapsto b \}, \{ z \mapsto c \} \) to the concept \( C \).

We simply get an absorption algorithm that generates these kind of binding candidates by changing Algorithm 8 and 9 such that \( A_{BP} \subseteq (A_1, A_2) \to_{\text{BP}} A' \) is added to \( T' \) in line 13 of Algorithm 8 instead of \( A_{BP} \subseteq (A_1, A_2) \to_{BP} A' \). Note, for the creation of binding candidates it is also useful to use the absorption with the backward chaining optimisation, because then the creation of binding candidates can be triggered with \( A_O \).

6 Variable Elimination Optimisations

It is often the case that the number of nominal schema variables can be reduced by rewriting the axiom. The basic idea is to replace unimportant nominal schemas with \( O \), the special atomic concept that is added to the label of every ABox individual. A nominal schema \( \{ x \} \) can obviously be replaced if \( \{ x \} \) occurs only once in the axiom and only in a completely absorbable position. Such an occurrence merely requires that there exists an ABox individual for \( x \), but it is not relevant to remember the individual, because \( x \) is only used once. Thus, we can use \( O \) instead of \( \{ x \} \). If \( \{ x \} \) is not in a completely absorbable position, then we have to be more careful. For example, \( \{ x \} \) cannot be replaced by \( O \) in the axioms \( A \subseteq \exists r.\{ x \} \) and \( A \subseteq \forall r.\{ x \} \cap \exists r.\top \).

We can sometimes also eliminate nominal schema variables that have more than one occurrence in the axiom. Therefore, we need an analogous definition of safe environments of a nominal schema as Krötzsch et al. [14]:

**Definition 14 (Safety).** Let \( C \) be a concept in a completely absorbable position, then an occurrence of a nominal schema \( \{ x \} \) is safe if it occurs in \( D \) with \( \text{Vars}(D) = \{ x \} \), \( \exists r.D \in \text{collectDisjuncts}(C, \text{true}) \) and \( \neg\{ o \} \in \text{collectDisjuncts}(C, \text{true}) \), where \( \{ o \} \) is a nominal or a nominal schema. Thus, \( C \) is a safe environment for such an occurrence of \( \{ x \} \). A nominal schema variable \( x \) is safe for an axiom \( W \) if every occurrence of \( \{ x \} \) is completely absorbable and at most one occurrence of \( \{ x \} \) is not safe.

A safe nominal schema variable \( x \) can be eliminated by rewriting the axiom. For example, the axiom

\[
\exists r.(\{ x \} \cap \exists a.\{ y \} \cap \exists w.\{ z \}) \cap \exists s.(\exists a.\{ y \} \cap \exists w.\{ z \}) \subseteq \exists c.\{ x \}
\]

has the two safe nominal schema variables \( y \) and \( z \), because \( \{ x \} \cap \exists r.\{ y \} \cap \exists w.\{ z \} \) is a safe environment for \( y \) and \( z \), there is only one other not safe occurrence of \( y \) and \( z \), and the left-hand side of the axiom is completely absorbable. We can now eliminate \( y \) and \( z \) by building the inverse path of existential restrictions from the safe occurrences of \( \{ y \} \) and \( \{ z \} \) to the nominal schema \( \{ x \} \) in the safe environment and appending this path together with \( O \) to the not safe occurrence of \( y \) and \( z \). Thus, the axiom is rewritten to

\[
\exists r.\{ x \} \cap \exists s.(\exists a.\{ O \cap \exists a^-.\{ x \} \} \cap \exists w.\{ O \cap \exists w^-.\{ x \}) \subseteq \exists c.\{ x \}
\]
with $x$ as the only remaining nominal schema variable.

7 Implementation and Evaluation

Our reasoning system Konclude is able to deal with $SROIQ$ knowledge bases and uses, besides many other optimisations, an absorption technique that is based on the one presented in Section 3. We have extended Konclude to $SROIQV$ by integrating (i) an upfront grounding of nominal schema axioms and (ii) tableau extensions with different optimisations for propagating variable mappings in order to support the presented nominal schema absorption in Section 4. The upfront grounding is not only used to compare our nominal schema absorption technique, but also to eliminate axioms with nominal schemas that cannot be absorbed at all. The upfront grounding is more efficient for concepts that are certainly used in the completion graph, because upfront grounded concepts can be better preprocessed and it is not necessary to dynamically extend the knowledge base during the construction of the completion graph for those concepts. This is especially useful for Konclude, since it supports the parallel processing of non-deterministic alternatives for which it would be necessary to synchronise the extensions of the knowledge base or to separately extend the knowledge base in each alternative.

Unfortunately, a straightforward implementation of the proposed propagation of variable mappings still bears the following sources of inefficiency:

- If a knowledge base contains an ABox individual $a$ that is connected to many other individuals by a role $r$, some mappings are propagated to $a$, and the propagation is continued over $r$, then these mappings are propagated to all such connected $r$-neighbours, even if the mappings are only required on a few of them.
- Dependency information for each propagated mapping has to be held separately in order to support dependency-directed backtracking.
- To create the correct dependencies, the variable mappings have to be joined on each node separately, even if we already have joined the same sets of variable mappings on other nodes.

We have, therefore, also implemented a variant of the propagation, where we create a representative for a set of variable mappings. We then propagate only these representatives and track the dependencies only for propagated representatives as in [19]. If a clash is discovered, then we extract and backtrack only the dependencies for those mappings, which are involved in the creation of the clash. In order to extract the relevant dependencies, we save for the representatives how they are composed from other representatives and variable mappings.

Furthermore, Konclude uses a batch processing mode for the variable mappings, i.e., Konclude tries to apply standard deterministic rules first and then the new rules that handle variable mappings are applied in the same order as the corresponding concepts and axioms are created in the absorption. Usually, this is a big advantage, because then many variable mappings can be handled together and the overhead of separate rule applications is minimized.

Our evaluation is primarily based on DL-safe rules to enable a comparison between the propagation of variable mappings that is integrated in Konclude and the DL rea-
Table 5. Ontology metrics

<table>
<thead>
<tr>
<th>Ontology</th>
<th>Expressiveness</th>
<th>Axioms</th>
<th>Classes</th>
<th>Properties</th>
<th>Individuals</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>UOBM\D</td>
<td>SHOIN</td>
<td>19093</td>
<td>69</td>
<td>36</td>
<td>25453</td>
<td>0</td>
</tr>
<tr>
<td>family\D</td>
<td>ALCCHOIN</td>
<td>212</td>
<td>19</td>
<td>16</td>
<td>23</td>
<td>12</td>
</tr>
<tr>
<td>ODGI\D</td>
<td>SHIN</td>
<td>2391</td>
<td>346</td>
<td>83</td>
<td>356</td>
<td>2</td>
</tr>
</tbody>
</table>

soners HermiT 1.3.7\(^3\) and Pellet 2.3.0 [18], which have dedicated rule support. To the best of our knowledge, these are the only reasoning systems that support DL-safe rules for such expressive ontologies. We have integrated a converter into Konclude, which transforms the DL-safe rules into nominal schema axioms. The conversion is straightforward, however, especially for nominal schema axioms that are obtained from DL-safe rules, it is often possible to eliminate variables as discussed in Section 6, which is intensively used by Konclude. For example, the DL-safe Rule (R1) is converted to the Nominal Schema Axiom (R1\'') and then \{x\} and \{z\} are replaced by \(O\), which results in Axiom (R1\'').

\[
\begin{align*}
\text{isFirendOf}(\?, x, \?, y), \text{like}(\?, x, \?, z), \text{like}(\?, y, \?, z) & \rightarrow \text{friendWithSameInterest}(\?, x, \?, y) \quad \text{(R1)} \\
\{x\} \sqcap \exists \text{like}.(\{z\} \sqcap \exists \text{like}^-.\{y\}) \sqcap \exists \text{isFirendOf}.\{y\} & \sqsubseteq \exists \text{friendWithSameInterest}.\{y\} \quad \text{(R1')} \\
\{O\} \sqcap \exists \text{like}.(\{O\} \sqcap \exists \text{like}^-.\{y\}) \sqcap \exists \text{isFirendOf}.\{y\} & \sqsubseteq \exists \text{friendWithSameInterest}.\{y\} \quad \text{(R1'\''})
\end{align*}
\]

All experiments were carried out on an Intel Core i7 940 quad core processor running at 2.93 GHz, however, all reasoners are restricted to use only one core for the computation. All results are the average of three separate runs. The execution of a test was aborted if either a reasoner required more than 24 hours or more than 10 GByte memory which is denoted by \textit{time} \textit{resp. mem} in the results.

7.1 UOBM-Benchmarks

For the evaluation, we have extended the University Ontology Benchmark (UOBM) [16] with DL-safe rules. We are only using the smallest UOBM ontology since many reasoning systems already require for this ontology a lot of memory as well as reasoning time if it is extended with rules. Furthermore, we have removed all data properties from the ontology because these are not yet supported by Konclude. We refer to this ontology as UOBM_1\D (cf. Table 5).

The hand-crafted DL-safe rules R1–R5 are depicted in Table 6, where the number of matches for each rule in the consistency check is shown in the column on the right side, i.e., how often a rule can be instantiated with different variable bindings. However, since the UOBM_1\D ontology is not completely deterministic, these numbers might vary between different executions and between different reasoners. All these rules contain at least one cycle, i.e., they do not have a tree-shaped form, and hence, these rules are not completely trivial.

The UOBM_1\D ontology without rules can be preprocessed by Konclude in 1.03 seconds and the corresponding consistency test requires only 1.09 seconds. Table 7

\[\text{http://www.hermit-reasoner.com}\]
Table 6. Hand-crafted DL-safe rules R1-R5 for the evaluation of UOBM₁\D

<table>
<thead>
<tr>
<th>Rule</th>
<th>DL-safe Rule</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>isFriendOf(?x, ?y), like(?x, ?z), like(?y, ?z) → friendWithSameInterest(?x, ?y)</td>
<td>4,037</td>
</tr>
<tr>
<td>R2</td>
<td>isFriendOf(?x, ?y), takesCourse(?x, ?z), takesCourse(?y, ?z) → friendWithSameCourse(?x, ?y)</td>
<td>82</td>
</tr>
<tr>
<td>R3</td>
<td>takesCourse(?x, ?z), takesCourse(?y, ?z), hasSameHomeTownWith(?x, ?y) → classmateWithSameHomeTown(?x, ?y)</td>
<td>940</td>
</tr>
<tr>
<td>R4</td>
<td>hasDoctoralDegreeFrom(?x, ?z), hasMasterDegreeFrom(?x, ?w), hasDoctoralDegreeFrom(?y, ?z), hasMasterDegreeFrom(?y, ?w), worksFor(?x, ?v), worksFor(?y, ?v), → workmateSameDegreeFrom(?x, ?y)</td>
<td>369</td>
</tr>
<tr>
<td>R5</td>
<td>isAdvisedBy(?x, ?z), isAdvisedBy(?y, ?z), like(?x, ?w), like(?y, ?w), like(?z, ?w) → personWithSameAdviserAllSameInterest(?x, ?y)</td>
<td>286</td>
</tr>
</tbody>
</table>

Table 7. Comparison of the increases in reasoning time of the consistency tests for UOBM₁\D extended by the rules R1–R5 between different techniques in seconds (additional preprocessing time in parentheses)

<table>
<thead>
<tr>
<th>Rule</th>
<th>upfront grounding</th>
<th>direct propagation without BP</th>
<th>direct propagation with BP</th>
<th>representative propagation without BP</th>
<th>representative propagation with BP</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>(10.99) mem</td>
<td>9.12</td>
<td>7.10</td>
<td>5.06</td>
<td>3.38</td>
</tr>
<tr>
<td>R2</td>
<td>(10.92) 4.05</td>
<td>3.33</td>
<td>2.33</td>
<td>2.13</td>
<td>2.11</td>
</tr>
<tr>
<td>R3</td>
<td>(13.33) 3.55</td>
<td>1.98</td>
<td>0.62</td>
<td>2.20</td>
<td>0.76</td>
</tr>
<tr>
<td>R4</td>
<td>(16.44) 0.30</td>
<td>1.08</td>
<td>0.09</td>
<td>1.06</td>
<td>0.07</td>
</tr>
<tr>
<td>R5</td>
<td>(time) –</td>
<td>1.87</td>
<td>0.50</td>
<td>1.80</td>
<td>0.43</td>
</tr>
</tbody>
</table>

shows the increases in reasoning time of the consistency tests for Konclude using different approaches and optimisations to handle the nominal schema axioms that are obtained by converting the DL-safe rules R1–R5 of Table 6. The additional required preprocessing time for the upfront grounding is shown in parentheses (column 2). Note, we show only the additional required times in order to facilitate the comparison, i.e., to get the actual required times, 1.03 and 1.09 have to be added to the preprocessing and reasoning times, respectively.

Clearly, the upfront grounding requires some additional preprocessing time for the grounding of the rules, but the majority of the time is required for further processing the grounded axioms (e.g., absorption, lexical normalisation, etc.). Note, the upfront grounding totally fails for R5, because the variable elimination can only eliminate two

Table 8. Comparison of the increases in reasoning time of the consistency tests for UOBM₁\D extended by the rules R1–R5 between Konclude, HermiT and Pellet in seconds

<table>
<thead>
<tr>
<th>Rule</th>
<th>Konclude 0.4.1</th>
<th>HermiT 1.3.7</th>
<th>Pellet 2.3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>3.38</td>
<td>31.46</td>
<td>6.33</td>
</tr>
<tr>
<td>R2</td>
<td>2.11</td>
<td>4.79</td>
<td>7.4</td>
</tr>
<tr>
<td>R3</td>
<td>0.46</td>
<td>1.67</td>
<td>142.25</td>
</tr>
<tr>
<td>R4</td>
<td>0.07</td>
<td>1.42</td>
<td>122.85</td>
</tr>
<tr>
<td>R5</td>
<td>0.43</td>
<td>28.41</td>
<td>mem</td>
</tr>
</tbody>
</table>
of four variables and the upfront grounding tries to generate 647, 855, 209 new axioms. Furthermore, the reasoning time for the upfront grounding is not as good as for the other approaches. This is due to the fact that the implicit propagation of variable mappings as concepts requires a lot of separate rule applications and each rule application has some overhead, e.g., the dependencies have to be managed correctly. The direct propagation of variable mappings (as presented in Section 4) and the propagation of representatives are shown without as well as with the back propagation optimisation (BP) in Table 7. Of course, the back propagation can only improve the reasoning time if it enables a significant reduction of the creation, combination and propagation of variable mappings. For example, the savings for R2 are very limited, because most of the students in the UOBM ontology take courses and also have friends and thus, almost all students match the conditions to be identified as a candidate for which the creation and propagation of variable mappings is necessary. Usually, the propagation of representatives further improves the reasoning time and, moreover, since the dependencies can be stored in a more compact way, it also saves a significant amount of memory.

Table 8 shows a comparison of the increases in reasoning time of the consistency tests for UOBM1\D extended by the DL-safe rules R1–R5 of Table 6 between Konclude, HermiT and Pellet. Again, the times for the consistency test of UOBM1\D without rules is not included to facilitate the comparison, i.e., to get the actual required times, it would be necessary to add for Konclude 1.09, for HermiT 23.24 and for Pellet 2.22 seconds. HermiT already requires a lot of time for the consistency test itself, which is possibly also a consequence of a delayed clausification of axioms and rules. However, since HermiT is not supporting complex roles in the body of rules, the consistency test might be incomplete for some rules. For example, the role hasSameHomeTownWith of rule R3 is transitive. Konclude uses the propagation of representatives combined with the back propagation optimisation and often tests the consistency for the UOBM1\D
In addition, we have converted the accompanying queries 1–15 from the University Ontology Benchmark [16] to the DL-safe rules Q1–Q15 (see Table 11). The comparison of the increases in reasoning time of the consistency tests for UOBM1\D extended by these rules is shown in Table 12. Konclude is dominating the other reasoners for nearly all rules, which is possible due to the variable elimination optimisation, which allows for absorbing the relatively simple and tree-shaped rules to ordinary concepts. HermiT can also handle many of these rules without significant performance losses, however, some of these rules again use transitive roles, wherefore HermiT might be incomplete (e.g., Q5, Q7, Q9, Q14). Pellet has more problems with the handling of the these rules and also requires significantly more memory. Pellet is even reaching the memory limit of 10 GBytes for rule Q7, whereas the other reasoners can process the consistency tests for all rules with less than one GByte.

Note, the consistency test is not an optimal way to compare different reasoners for their rule processing mechanism since there are also many other optimisations that influence the required reasoning time. However, this is even more problematic for more high level reasoning tasks such as instance retrieval or classification. For example, Konclude needs additional memory for the reasoning with the rules (less than 10 MBytes for Q7), whereas the other reasoners can process the consistency tests for all rules with less than one GByte. However, this is even more problematic for more high level reasoning tasks such as instance retrieval or classification.

<table>
<thead>
<tr>
<th>Name</th>
<th>DL-safe Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>UndergraduateStudent(?x), takesCourse(?x, D0.U0/Course0) → Q1(?x)</td>
</tr>
<tr>
<td>Q2</td>
<td>Employer(?x) → Q2(?x)</td>
</tr>
<tr>
<td>Q3</td>
<td>Student(?x), isMemberOf(?x, D0.U0) → Q3(?x)</td>
</tr>
<tr>
<td>Q4</td>
<td>Publication(?x), publicationAuthor(?x, ?y), Faculty(?y), isMemberOf(?y, D0.U0) → Q4(?x)</td>
</tr>
<tr>
<td>Q5</td>
<td>ResearchGroup(?x), subOrganizationOf(?x, U0) → Q5(?x)</td>
</tr>
<tr>
<td>Q6</td>
<td>Person(?x), hasAlumnus(U0, ?x) → Q6(?x)</td>
</tr>
<tr>
<td>Q7</td>
<td>Person(?x), hasSameHomeTownWith(?x, D0.U0/FullProfessor0) → Q7(?x)</td>
</tr>
<tr>
<td>Q8</td>
<td>SportsLover(?x), hasMember(D0.U0, ?x) → Q8(?x)</td>
</tr>
<tr>
<td>Q9</td>
<td>GraduateCourse(?x), isTaughtBy(?x, ?y), isMemberOf(?y, ?z), subOrganizationOf(?z, U0) → Q9(?x)</td>
</tr>
<tr>
<td>Q10</td>
<td>isFriendOf(?x, D0.U0/FullProfessor0) → Q10(?x)</td>
</tr>
<tr>
<td>Q11</td>
<td>Person(?x), like(?x, ?y), Chair(?z), isHeadOf(?z, D0.U0), like(?z, ?y) → Q11(?x)</td>
</tr>
<tr>
<td>Q12</td>
<td>Student(?x), takesCourse(?x, ?y), isTaughtBy(?y, D0.U0/FullProfessor0) → Q12(?x)</td>
</tr>
<tr>
<td>Q13</td>
<td>PeopleWithHobby(?x), isMemberOf(?x, D0.U0) → Q13(?x)</td>
</tr>
<tr>
<td>Q14</td>
<td>Woman(?x), Student(?x), isMemberOf(?x, ?y), subOrganizationOf(?y, U0) → Q14(?x)</td>
</tr>
<tr>
<td>Q15</td>
<td>PeopleWithManyHobbies(?x), isMemberOf(?x, D0.U0) → Q15(?x)</td>
</tr>
</tbody>
</table>

Table 11. DL-safe rules Q1–Q15 obtained from UOBM queries
Table 12. Comparison of the increases in reasoning time of the consistency tests for UOBM\(\text{1}\)\(\text{D}\) extended by the rules Q1–Q15 between Konclude, HermiT and Pellet in seconds

<table>
<thead>
<tr>
<th>Rule</th>
<th>Konclude 0.4.1</th>
<th>HermiT 1.3.7</th>
<th>Pellet 2.3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>(\leq 0.1)</td>
<td>(\leq 0.1)</td>
<td>3.9</td>
</tr>
<tr>
<td>Q2</td>
<td>(\leq 0.1)</td>
<td>(\leq 0.1)</td>
<td>720.0</td>
</tr>
<tr>
<td>Q3</td>
<td>(\leq 0.1)</td>
<td>(\leq 0.1)</td>
<td>2.5</td>
</tr>
<tr>
<td>Q4</td>
<td>(\leq 0.1)</td>
<td>(\leq 0.1)</td>
<td>2.5</td>
</tr>
<tr>
<td>Q5</td>
<td>(\leq 0.1)</td>
<td>(\leq 0.1)</td>
<td>12.4</td>
</tr>
<tr>
<td>Q6</td>
<td>(\leq 0.1)</td>
<td>(\leq 0.1)</td>
<td>2.0</td>
</tr>
<tr>
<td>Q7</td>
<td>(\leq 0.1)</td>
<td>(\leq 0.1)</td>
<td>mem</td>
</tr>
<tr>
<td>Q8</td>
<td>(\leq 0.1)</td>
<td>(\leq 0.1)</td>
<td>113.4</td>
</tr>
<tr>
<td>Q9</td>
<td>0.2</td>
<td>0.9</td>
<td>460.9</td>
</tr>
<tr>
<td>Q10</td>
<td>(\leq 0.1)</td>
<td>1.5</td>
<td>5.5</td>
</tr>
<tr>
<td>Q11</td>
<td>0.1</td>
<td>26.3</td>
<td>12.6</td>
</tr>
<tr>
<td>Q12</td>
<td>0.2</td>
<td>1.4</td>
<td>4.6</td>
</tr>
<tr>
<td>Q13</td>
<td>(\leq 0.1)</td>
<td>0.6</td>
<td>256.9</td>
</tr>
<tr>
<td>Q14</td>
<td>0.3</td>
<td>1.2</td>
<td>2730.4</td>
</tr>
<tr>
<td>Q15</td>
<td>1.2</td>
<td>(\leq 0.1)</td>
<td>2.9</td>
</tr>
</tbody>
</table>

conclude needs only 0.2 seconds to classify the UOBM\(\text{1}\)\(\text{D}\) ontology after the consistency check due to sophisticated caching mechanisms, whereas Pellet requires 9.9 and HermiT 271.7 seconds.

7.2 OpenRuleBench-Benchmarks

We adapted tests from OpenRuleBench [15] to compare the reasoning systems for some specific test cases. Basically, the LargeJoin test case gives an impression of the capability of handling binary joins. Therefore, the reasoners have to create new role instantiations for the roles \(c1\), \(b1\), \(b2\) and finally \(a\) between ABox individuals based on the following non-recursive, tree-shaped rules:

\[
d1(?x, ?y), d2(?y, ?z) \rightarrow c1(?x, ?z) \\
c1(?x, ?y), c2(?y, ?z) \rightarrow b1(?x, ?z) \\
c3(?x, ?y), c4(?y, ?z) \rightarrow b2(?x, ?z) \\
b1(?x, ?y), b2(?y, ?z) \rightarrow a(?x, ?z)
\]

The assertions for the base roles \(d1\), \(d2\), \(c2\), \(c3\) and \(c4\) were randomly generated. Table 13 shows the comparison of the reasoning times of the consistency tests between Konclude, HermiT and Pellet for different data sizes, i.e., the number of individuals is shown in the first column and the number of randomly added assertions for each base role is shown in the second column. Clearly, HermiT is dominating the other systems for this test case. Again, Konclude uses the back propagation optimisation and the propagation of representatives. However, the rules are very simple, wherefore the benefits of these optimisations are limited. Furthermore, Konclude does not directly add the implied role instantiation, but adds an existential restriction for the corresponding role, which is processed in a separate step and is, therefore, more costly.

Table 14 shows the comparison of the memory consumption of the consistency tests for the LargeJoin test case between Konclude, HermiT and Pellet. Also for this test case Konclude has a significant higher memory consumption than HermiT, which is not very
Table 13. Comparison of the reasoning times of the consistency tests for the LargeJoin test case between Konclude, HermiT and Pellet in seconds

<table>
<thead>
<tr>
<th># individuals</th>
<th># assertions per base role</th>
<th>Konclude 0.4.1</th>
<th>HermiT 1.3.7</th>
<th>Pellet 2.3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>50,000</td>
<td>10,000</td>
<td>1.2</td>
<td>1.0</td>
<td>3.7</td>
</tr>
<tr>
<td>50,000</td>
<td>20,000</td>
<td>3.4</td>
<td>2.0</td>
<td>6.2</td>
</tr>
<tr>
<td>50,000</td>
<td>30,000</td>
<td>6.9</td>
<td>3.4</td>
<td>10.5</td>
</tr>
<tr>
<td>100,000</td>
<td>20,000</td>
<td>3.5</td>
<td>1.6</td>
<td>7.0</td>
</tr>
<tr>
<td>100,000</td>
<td>40,000</td>
<td>11.1</td>
<td>3.8</td>
<td>25.4</td>
</tr>
<tr>
<td>100,000</td>
<td>60,000</td>
<td>23.6</td>
<td>10.5</td>
<td>97.1</td>
</tr>
</tbody>
</table>

Table 14. Comparison of the memory consumption of the consistency tests for the LargeJoin test case between Konclude, HermiT and Pellet in MBytes

<table>
<thead>
<tr>
<th># individuals</th>
<th># assertions per base role</th>
<th>Konclude 0.4.1</th>
<th>HermiT 1.3.7</th>
<th>Pellet 2.3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>50,000</td>
<td>10,000</td>
<td>507</td>
<td>985</td>
<td>1452</td>
</tr>
<tr>
<td>50,000</td>
<td>20,000</td>
<td>847</td>
<td>1240</td>
<td>1739</td>
</tr>
<tr>
<td>50,000</td>
<td>30,000</td>
<td>1480</td>
<td>1453</td>
<td>2548</td>
</tr>
<tr>
<td>100,000</td>
<td>20,000</td>
<td>1129</td>
<td>1175</td>
<td>2023</td>
</tr>
<tr>
<td>100,000</td>
<td>40,000</td>
<td>2380</td>
<td>1358</td>
<td>3535</td>
</tr>
<tr>
<td>100,000</td>
<td>60,000</td>
<td>4304</td>
<td>1940</td>
<td>7400</td>
</tr>
</tbody>
</table>

surprising, because Konclude saves intermediate results (e.g., the propagated variable mappings with dependency information) and also requires some additional memory for the grounded existential restrictions.

LargeJoin-M is a variant of the test case LargeJoin, where only the final role a has to be instantiated by the reasoners, which is achieved by merging the rules of the LargeJoin test case together into the following rule:

\[
d1(x, y_1), d2(y_1, y_2), c2(y_2, y_3), c3(y_3, y_4), c4(y_4, z) \rightarrow a(x, z).
\]

Table 15 and 16 shows the comparison of the reasoning times and memory consumption, respectively, of the consistency tests for LargeJoin-M with the same test data as for LargeJoin. Now, the back propagation optimisation significantly improves the reasoning time and memory consumption for Konclude and there are also some improvements for Pellet, which is possibly due to the reduced number of role instantiations that have to be created. In contrast, HermiT performs worse for bigger data sizes than for the LargeJoin test case.

The transitive closure test TransClos-T, which is also adapted from OpenRuleBench [15], demonstrates the performance for a simple transitive/recursive problem. All individuals in the test are connected in a big cycle with the transitive role \( par \), and the rule

\[
par(x, y) \rightarrow tc(x, y)
\]

is used to enforce that the transitive closure is explicitly represented with the role \( tc \). Additionally, some random assertions for the role \( par \) were added to make the test case not completely straightforward. Table 17 shows the comparison of the reasoning times of the consistency tests for the test case TransClos-T between Konclude and Pellet. HermiT does not consider the transitivity in the body of the rule and is, therefore, omitted in this comparison. The transitivity can, however, be simulated for this test case by using
Table 15. Comparison of the reasoning times of the consistency tests for the LargeJoin-M test case between Konclude, HermiT and Pellet in seconds

<table>
<thead>
<tr>
<th># individuals</th>
<th># assertions per role</th>
<th>Konclude 0.4.1</th>
<th>HermiT 1.3.7</th>
<th>Pellet 2.3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>50,000</td>
<td>10,000</td>
<td>0.7</td>
<td>0.9</td>
<td>1.6</td>
</tr>
<tr>
<td>50,000</td>
<td>20,000</td>
<td>1.1</td>
<td>1.4</td>
<td>4.2</td>
</tr>
<tr>
<td>50,000</td>
<td>30,000</td>
<td>2.6</td>
<td>2.6</td>
<td>7.6</td>
</tr>
<tr>
<td>100,000</td>
<td>20,000</td>
<td>1.4</td>
<td>1.6</td>
<td>5.7</td>
</tr>
<tr>
<td>100,000</td>
<td>40,000</td>
<td>5.1</td>
<td>5.3</td>
<td>18.0</td>
</tr>
<tr>
<td>100,000</td>
<td>60,000</td>
<td>11.8</td>
<td>20.4</td>
<td>94.9</td>
</tr>
</tbody>
</table>

Table 16. Comparison of the memory consumption of the consistency tests for the LargeJoin-M test case between Konclude, HermiT and Pellet in MBytes

<table>
<thead>
<tr>
<th># individuals</th>
<th># assertions per role</th>
<th>Konclude 0.4.1</th>
<th>HermiT 1.3.7</th>
<th>Pellet 2.3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>50,000</td>
<td>10,000</td>
<td>342</td>
<td>974</td>
<td>1361</td>
</tr>
<tr>
<td>50,000</td>
<td>20,000</td>
<td>483</td>
<td>1248</td>
<td>1526</td>
</tr>
<tr>
<td>50,000</td>
<td>30,000</td>
<td>702</td>
<td>1289</td>
<td>1623</td>
</tr>
<tr>
<td>100,000</td>
<td>20,000</td>
<td>723</td>
<td>1170</td>
<td>1719</td>
</tr>
<tr>
<td>100,000</td>
<td>40,000</td>
<td>1405</td>
<td>1393</td>
<td>3435</td>
</tr>
<tr>
<td>100,000</td>
<td>60,000</td>
<td>2569</td>
<td>1602</td>
<td>7114</td>
</tr>
</tbody>
</table>

The recursive rule
\[ tc(?x, ?y), \text{par}(?y, ?z) \rightarrow tc(?x, ?z), \]
which is denoted by TransClos-R, i.e., TransClos-R is a variant of TransClos-T in which this rule is used instead of the transitivity axiom for the role \( \text{par} \). Thus, this test case is also correctly handled by HermiT. The comparison of the reasoning times of the consistency tests for this test case is shown in Table 18. Konclude requires more or less the same amount of resources for both test cases. However, there are some differences in the propagation, wherefore the memory consumption for the test case TransClos-T is slightly higher. Although Konclude and HermiT can easily process TransClos-R with less than one GByte and approximately the same amount of reasoning time, this test case is much more problematic for Pellet, which suddenly requires more than 10 GBytes for TransClos-R compared to less than 2 GBytes for TransClos-T.

Table 17. Comparison of the reasoning times of the consistency tests for the TransClos-T test case between Konclude and Pellet in seconds

<table>
<thead>
<tr>
<th># individuals</th>
<th># random assertions</th>
<th>Konclude 0.4.1</th>
<th>Pellet 2.3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>100</td>
<td>1.2</td>
<td>8021.1</td>
</tr>
<tr>
<td>500</td>
<td>200</td>
<td>1.8</td>
<td>7963.6</td>
</tr>
<tr>
<td>1,000</td>
<td>200</td>
<td>6.7</td>
<td>time</td>
</tr>
<tr>
<td>1,000</td>
<td>400</td>
<td>8.2</td>
<td>time</td>
</tr>
</tbody>
</table>
Table 18. Comparison of the reasoning times of the consistency tests for the TransClos-R test case between Konclude, HermiT and Pellet in seconds

<table>
<thead>
<tr>
<th># individuals</th>
<th># random assertions</th>
<th>Konclude 0.4.1</th>
<th>HermiT 1.3.7</th>
<th>Pellet 2.3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>100</td>
<td>2.1</td>
<td>1.0</td>
<td>mem</td>
</tr>
<tr>
<td>500</td>
<td>200</td>
<td>1.1</td>
<td>1.0</td>
<td>mem</td>
</tr>
<tr>
<td>1,000</td>
<td>200</td>
<td>5.1</td>
<td>4.7</td>
<td>mem</td>
</tr>
<tr>
<td>1,000</td>
<td>400</td>
<td>3.3</td>
<td>5.1</td>
<td>mem</td>
</tr>
</tbody>
</table>

7.3 Benchmarks for Ontologies with Rules

We use family\D_{100} and ODGI\D_{100} to compare the reasoning times for ontologies with already integrated rules. The family\D ontology (see Table 5) is obtained from the family.swrl.owl demo ontology in the Protégé Ontology Library\(^4\) by removing all data properties and rules with SWRL Built-Ins. In order to get an interesting benchmark size, we created family\D_{100} from family\D by adding 100 copies of the ABox with renamed non-nominal individuals. Analogously, we obtained ODGI\D (see Table 5) and ODGI\D_{100} from the ontology for disease genetic investigation (ODGI) from the NCBO BioPortal\(^5\).

Table 19 shows the comparison of the reasoning times of the consistency tests for these ontologies with and without rules. Although the family\D_{100} ontology contains the rule

\[
\text{Person(?y), hasChild(?y, ?x), hasChild(?y, ?z), ?x \neq ?z \to hasSibling(?x, ?z),}
\]

which uses the atom \(?x \neq ?z\) in the body of the rule that states that two individuals have to be different, the reasoning time of HermiT is hardly affected. Usually, such rules cannot completely absorbed and, therefore, the application of such rules lead to non-determinism. Due to the fact that this rule has to be applied quite often, the reasoning with the family\D_{100} ontology is highly non-deterministic. This seems to be more problematic for Konclude and Pellet, possibly due to their different dependency management. However, HermiT performs worse for ODGI\D_{100} than Konclude and Pellet. The ODGI\D_{100} ontology also contains transitive roles, however, they are not used in the rule bodies, and hence, also the results for HermiT are complete. Analogously, Table 20 shows the comparison of the memory consumption of the consistency tests for these ontologies with and without rules.

\(^4\) http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library
\(^5\) http://bioportal.bioontology.org/ontologies/1086

Table 19. Comparison of the reasoning times of the consistency tests for ontologies with DL-safe rules between Konclude, HermiT and Pellet in seconds

<table>
<thead>
<tr>
<th>Ontology</th>
<th>Konclude 0.4.1</th>
<th>HermiT 1.3.7</th>
<th>Pellet 2.3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>family\D_{100}</td>
<td>0.2</td>
<td>0.6</td>
<td>1.3</td>
</tr>
<tr>
<td>family\D_{100}</td>
<td>2.5</td>
<td>1.3</td>
<td>mem</td>
</tr>
<tr>
<td>ODGI\D_{100}</td>
<td>11.4</td>
<td>367.6</td>
<td>13.3</td>
</tr>
<tr>
<td>ODGI\D_{100}</td>
<td>13.6</td>
<td>467.4</td>
<td>17.8</td>
</tr>
</tbody>
</table>
Table 20. Comparison of the memory consumption of the consistency tests for ontologies with DL-safe rules between Konclude, HermiT and Pellet in MBytes

<table>
<thead>
<tr>
<th>Ontology</th>
<th>Konclude 0.4.1</th>
<th>HermiT 1.3.7</th>
<th>Pellet 2.3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>family\D_{100} without rules</td>
<td>147</td>
<td>473</td>
<td>771</td>
</tr>
<tr>
<td>family\D_{100} with rules</td>
<td>954</td>
<td>744</td>
<td>mem</td>
</tr>
<tr>
<td>ODGI\D_{100} without rules</td>
<td>704</td>
<td>1302</td>
<td>1658</td>
</tr>
<tr>
<td>ODGI\D_{100} with rules</td>
<td>884</td>
<td>1456</td>
<td>1852</td>
</tr>
</tbody>
</table>

7.4 Benchmarks beyond Rules

Unfortunately, there exists no test suite for nominal schemas and, to the best of our knowledge, there are also no other reasoners with nominal schema support. Therefore, the evaluation of nominal schema axioms, which are more expressive than DL-safe rules, is less conclusive, since the results cannot be compared to the results of other systems. Also, the comparison to the upfront grounding is often not very interesting, since the performance of the upfront grounding mainly depends on the additional required preprocessing time, which primarily relies on the number of nominal schema variables and the number of ABox individuals. Hence, the performance of the upfront grounding and the performance gain with the absorption and the propagation of variable mappings is more or less predictable and, therefore, it is easily possible to create tests, where the nominal schema absorption approach performs as much better as desired. Note, nominal schema axioms that are more expressive than DL-safe rules often reduce the absorbable amount on the left-hand side of the axiom. This is, however, not necessarily problematic, because the nominal schemas can often also be absorbed in other parts of the axiom. For instance, the nominal schema axiom

\[ \{x \} \sqcap \exists \text{like}.\{y\} \sqcap \forall \text{isFriendOf}.\exists \text{like}.\{y\} \sqsubseteq \text{HasInterestAndAllFriendsSameInterest} \]

identifies for the UOBM\_\D ontology all ABox individuals, which have an interest and the friends of these individuals have at least also the same interest. Obviously, the universal restriction \( \forall \text{isFriendOf}.\exists \text{like}.\{y\} \) on the left-hand side cannot be absorbed. However, \( \{y\} \) also occurs in the absorbable existential restriction \( \exists \text{like}.\{y\} \) in an absorbable position, wherefore the nominal schema absorption approach is still able to significantly reduce the grounding effort. For UOBM\_\D extended with this nominal schema axiom, Konclude requires for the consistency test with the upfront grounding 6.1 seconds of additional preprocessing time and 3.4 seconds of additional reasoning time, whereas the nominal schema absorption approach only increases the reasoning time by 0.9 seconds.

Of course, there still exists nominal schema axioms with nominal schemas that only occur in non-absorbable positions. For example, the axiom

\[ \{x\} \sqcap \exists \text{like}.\{y\} \sqcap \forall \text{isFriendOf}(\{z\} \sqcap \exists \text{like}.\{y\}) \sqsubseteq \text{HasInterestAndAllFriendsSameInterest} \]

extends the previous nominal schema axiom with the nominal schema \( \{z\} \), which further enforces that all friends also have to be known ABox individuals. For this axiom, the grounding concept that is created by the absorption has to handle the disjunction
HasInterestAndAllFriendsSameInterest ⊔ \exists isFriendOf(¬{z} ⊔ ¬\forall like.¬{y}) and, since only bindings for the variable \( y \) are created and propagated, the grounding rule has to complete the variable mappings to the variable \( z \) by combining them with every ABox individual. As a consequence, the grounding rule adds quite a lot of different disjunctions and this for many nodes in the completion graph, wherefore Konclude with the nominal schema absorption is running out of memory. However, this would obviously also be the case with the upfront grounding if the preprocessing for the upfront grounding were able to finish within the time limit, which fails analogously to the processing of rule R5, since only the nominal schema variable \( x \) can be eliminated.

In principle, it would be possible to use a more sophisticated axiom rewriting in order to improve the absorption and, as a consequence, also the overall performance for the reasoning with such axioms. For example, the existentially required connection between \( \{z\} \) and \( \{y\} \) over the role \( like \) within the universal restriction \( \forall isFriendOf.(\{z\} ⊔ \exists like.\{y\}) \) can also be expressed in the inverse direction within the absorbable existential restriction \( \exists like.\{y\} \), which results in the axiom:

\[
\{x\} ⊔ \exists like.(\{y\} ⊔ \exists like^-.\{z\}) ⊔ \forall isFriendOf.(\{z\} ⊔ \exists like.\{y\}) ⊑ HasInterestAndAllFriendsSameInterest.
\]

Although now also \( \{z\} \) can be partially absorbed, Konclude still requires for the consistency test of UOBM1\(\setminus D \) extended by this nominal schema axiom 35.9 seconds and 8.2 GBytes due to the huge number of individuals that are found for \( \{z\} \). Unsurprisingly, the intensive use of nominal schemas in not completely absorbable constructs and the frequent grounding of many such not absorbed concepts results in a lot of work for the reasoning system.

8 Conclusions

We have significantly improved the reasoning performance for nominal schemas with (i) an extended absorption algorithm as well as (ii) slight modifications of the standard tableau calculus. The resulting calculus creates bindings for nominal schema variables and is able to propagate them through the completion graph in order to use these bindings to ground the remaining and non-absorbable part of the nominal schema axioms. As a consequence, our approach allows for “collecting” the bindings for those nominal schema axioms that have to be grounded and considered for a specific node in the completion graph. We have shown the correctness of the nominal schema absorption and, moreover, we have also presented techniques for further optimisations.

The approach only improves the handling of “absorbable” axioms, but, to the best of our knowledge, this restriction is satisfied for the majority of all nominal schema axioms that are used in practical ontologies. The presented techniques have been integrated into the novel reasoning system Konclude, which is now able to handle SROIQV knowledge bases, and the empirical evaluation, which is primarily based on DL-safe rules, shows that our approach performs well even when compared to other well-known DL reasoners with dedicated rule support. In particular, the performance for rules with complex roles is significantly better than in other reasoning systems for more expressive DLs.
The presented techniques are also interesting for the extension of existing tableau-based DL reasoners to ordinary DL-safe rules, since they allow a direct integration of the support of DL-safe rules into the tableau algorithm, whereby an additional/separate inference mechanism for the rules is not required.

Acknowledgements

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

References

Liste der bisher erschienenen Ulmer Informatik-Berichte
Einige davon sind per FTP von ftp.informatik.uni-ulm.de erhältlich
Die mit * markierten Berichte sind vergriffen

List of technical reports published by the University of Ulm
Some of them are available by FTP from ftp.informatik.uni-ulm.de
Reports marked with * are out of print

<table>
<thead>
<tr>
<th>No.</th>
<th>Author(s)</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>91-01</td>
<td>Ker-I Ko, P. Orponen, U. Schöning, O. Watanabe</td>
<td>Instance Complexity</td>
</tr>
<tr>
<td>91-02*</td>
<td>K. Gladitz, H. Fassbender, H. Vogler</td>
<td>Compiler-Based Implementation of Syntax-Directed Functional Programming</td>
</tr>
<tr>
<td>91-03*</td>
<td>Alfons Geser</td>
<td>Relative Termination</td>
</tr>
<tr>
<td>91-04*</td>
<td>J. Köbler, U. Schöning, J. Toran</td>
<td>Graph Isomorphism is low for PP</td>
</tr>
<tr>
<td>91-05</td>
<td>Johannes Köbler, Thomas Thierauf</td>
<td>Complexity Restricted Advice Functions</td>
</tr>
<tr>
<td>91-06*</td>
<td>Uwe Schöning</td>
<td>Recent Highlights in Structural Complexity Theory</td>
</tr>
<tr>
<td>91-07*</td>
<td>F. Green, J. Köbler, J. Toran</td>
<td>The Power of Middle Bit</td>
</tr>
<tr>
<td>91-08*</td>
<td>V. Arvind, Y. Han, L. Hamachandra, J. Köbler, A. Lozano, M. Mundhenk, A. Ogiwara, U. Schöning, R. Silvestri, T. Thierauf</td>
<td>Reductions for Sets of Low Information Content</td>
</tr>
<tr>
<td>92-01*</td>
<td>Vikraman Arvind, Johannes Köbler, Martin Mundhenk</td>
<td>On Bounded Truth-Table and Conjunctive Reductions to Sparse and Tally Sets</td>
</tr>
<tr>
<td>92-02*</td>
<td>Thomas Noll, Heiko Vogler</td>
<td>Top-down Parsing with Simultaneous Evaluation of Noncircular Attribute Grammars</td>
</tr>
<tr>
<td>92-03</td>
<td>Fakultät für Informatik</td>
<td>17. Workshop über Komplexitätstheorie, effiziente Algorithmen und Datenstrukturen</td>
</tr>
<tr>
<td>92-04*</td>
<td>V. Arvind, J. Köbler, M. Mundhenk</td>
<td>Lowness and the Complexity of Sparse and Tally Descriptions</td>
</tr>
<tr>
<td>92-05*</td>
<td>Johannes Köbler</td>
<td>Locating P/poly Optimally in the Extended Low Hierarchy</td>
</tr>
<tr>
<td>92-06*</td>
<td>Armin Kühnemann, Heiko Vogler</td>
<td>Synthesized and inherited functions -a new computational model for syntax-directed semantics</td>
</tr>
<tr>
<td>92-07*</td>
<td>Heinz Fassbender, Heiko Vogler</td>
<td>A Universal Unification Algorithm Based on Unification-Driven Leftmost Outermost Narrowing</td>
</tr>
</tbody>
</table>
92-08* Uwe Schöning
On Random Reductions from Sparse Sets to Tally Sets

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

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

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

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

92-13 Alfons Geser
On a monotonic semantic path ordering

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

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

The COCOON Object Model

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

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

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

93-06 Klaus Gaßner
Rechnerunterstützung für die konzeptuelle Modellierung

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

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

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

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

94-04 Heinz Faßbender, Heiko Vogler, Andrea Wedel
Implementation of a Deterministic Partial E-Unification Algorithm for Macro Tree Transducers
94-05  V. Arvind, J. Köbler, R. Schuler  
On Helping and Interactive Proof Systems

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

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

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

Construction and Deduction Methods for the Formal Development of Software

94-10  Axel Dold  
Formalisierung schematischer Algorithmen

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

94-12  Rainer Schuler  
On Average Polynomial Time

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

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

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

94-16  Robert Regn  
Verteilte Unix-Betriebssysteme

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

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

95-01  Oleg Verbitsky  
On the Largest Common Subgraph Problem

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

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

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

95-05  Klaus Achatz, Wolfram Schulte  
Architecture Independent Massive Parallelization of Divide-And-Conquer Algorithms
<table>
<thead>
<tr>
<th>Paper ID</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>95-06</td>
<td>Christoph Karg, Rainer Schuler</td>
<td>Structure in Average Case Complexity</td>
</tr>
<tr>
<td>95-07</td>
<td>P. Dadam, K. Kuhn, M. Reichert, T. Beuter, M. Nathe</td>
<td>ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen</td>
</tr>
<tr>
<td>95-08</td>
<td>Jürgen Kehrer, Peter Schulthess</td>
<td>Aufbereitung von gescannten Röntgenbildern zur filmlosen Diagnostik</td>
</tr>
<tr>
<td>95-09</td>
<td>Hans-Jörg Burtschick, Wolfgang Lindner</td>
<td>On Sets Turing Reducible to P-Selective Sets</td>
</tr>
<tr>
<td>95-10</td>
<td>Boris Hartmann</td>
<td>Berücksichtigung lokaler Randbedingung bei globaler Zieloptimierung mit neuronalen Netzen am Beispiel Truck Backer-Upper</td>
</tr>
<tr>
<td>95-11</td>
<td>Thomas Beuter, Peter Dadam</td>
<td>Prinzipien der Replikationskontrolle in verteilten Systemen</td>
</tr>
<tr>
<td>95-12</td>
<td>Klaus Achatz, Wolfram Schulte</td>
<td>Massive Parallelization of Divide-and-Conquer Algorithms over Powerlists</td>
</tr>
<tr>
<td>95-13</td>
<td>Andrea Mößle, Heiko Vogler</td>
<td>Efficient Call-by-value Evaluation Strategy of Primitive Recursive Program Schemes</td>
</tr>
<tr>
<td>96-01</td>
<td>Ercüment Canver, Jan-Tecker Gayen, Adam Moik</td>
<td>Formale Entwicklung der Steuerungssoftware für eine elektrisch ortsbediente Weiche mit VSE</td>
</tr>
<tr>
<td>96-02</td>
<td>Bernhard Nebel</td>
<td>Solving Hard Qualitative Temporal Reasoning Problems: Evaluating the Efficiency of Using the ORD-Horn Class</td>
</tr>
<tr>
<td>96-03</td>
<td>Ton Vullinghs, Wolfram Schulte, Thilo Schwinn</td>
<td>An Introduction to TkGofer</td>
</tr>
<tr>
<td>96-04</td>
<td>Thomas Beuter, Peter Dadam</td>
<td>Anwendungsspezifische Anforderungen an Workflow-Mangement-Systeme am Beispiel der Domäne Concurrent-Engineering</td>
</tr>
<tr>
<td>96-05</td>
<td>Gerhard Schellhorn, Wolfgang Ahrendt</td>
<td>Verification of a Prolog Compiler - First Steps with KIV</td>
</tr>
<tr>
<td>96-06</td>
<td>Manindra Agrawal, Thomas Thierauf</td>
<td>Satisfiability Problems</td>
</tr>
<tr>
<td>96-07</td>
<td>Vikraman Arvind, Jacobo Torán</td>
<td>A nonadaptive NC Checker for Permutation Group Intersection</td>
</tr>
<tr>
<td>96-08</td>
<td>David Cyrluk, Oliver Möller, Harald Rueß</td>
<td>An Efficient Decision Procedure for a Theory of Fix-Sized Bitvectors with Composition and Extraction</td>
</tr>
</tbody>
</table>
96-09 Bernd Biechele, Dietmar Ernst, Frank Houdek, Joachim Schmid, Wolfram Schulte
Erfahrungen bei der Modellierung eingebetteter Systeme mit verschiedenen SA/RT-Ansätzen

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

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

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

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

97-01 Jochen Messner
Pattern Matching in Trace Monoids

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

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

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

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

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

97-07 Manfred Reichert, Peter Dadam
ADEPTflex - Supporting Dynamic Changes of Workflows Without Losing Control

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

97-09 Christian Heinlein
Grundlagen von Interaktionsausdrücken

97-10 Christian Heinlein
Graphische Repräsentation von Interaktionsausdrücken

97-11 Christian Heinlein
Sprachtheoretische Semantik von Interaktionsausdrücken
Gerhard Schellhorn, Wolfgang Reif
Proving Properties of Finite Enumerations: A Problem Set for Automated Theorem Provers

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

Wolfgang Reif, Gerhard Schellhorn
Theorem Proving in Large Theories

Thomas Wennemers
Asymptotic recurrent neural networks with random connections

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

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

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

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

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

Marko Luther, Martin Strecker
A guided tour through Typelab

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

Ercüment Canver
Formal Verification of a Coordinated Atomic Action Based Design

Andreas Küchler
On the Correspondence between Neural Folding Architectures and Tree Automata

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

Thomas Wennemers
Synfire Graphs: From Spike Patterns to Automata of Spiking Neurons

Thomas Bauer, Peter Dadam
Variable Migration von Workflows in ADEPT

Heiko Neumann, Wolfgang Sepp
Recurrent V1–V2 Interaction in Early Visual Boundary Processing
98-11 Frank Houdek, Dietmar Ernst, Thilo Schwinn
Prüfen von C–Code und Statmate/Matlab–Spezifikationen: Ein Experiment

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

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

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

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

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

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

99-03 Uwe Schöning
On the Complexity of Constraint Satisfaction

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

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

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

99-07 Peter Dadam, Manfred Reichert

99-08 Vikraman Arvind, Johannes Köbler
Graph Isomorphism is Low for ZPPNP and other Lowness results

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

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

2000-03 Gregory Baratoff, Christian Toepfer, Heiko Neumann
Combined space-variant maps for optical flow based navigation
2000-04  Wolfgang Gehring
Ein Rahmenwerk zur Einführung von Leistungspunktsystemen

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

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

2000-07  Gerhard Schellhorn, Wolfgang Reif (eds.)

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

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

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

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

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

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

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

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

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

2002-02  Walter Guttmann
Deriving an Applicative Heapsort Algorithm

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

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

2003-02  Stefanie Rinderle, Manfred Reichert, Peter Dadam
Supporting Workflow Schema Evolution By Efficient Compliance Checks
2003-03  Christian Heinlein
Safely Extending Procedure Types to Allow Nested Procedures as Values

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

2003-05  Christian Heinlein
Dynamic Class Methods in Java

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

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

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

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

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

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

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

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

2005-03  Walter Guttmann, Markus Maucher
Constrained Ordering

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

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

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

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

2008-01  H.A. Kestler, J. Messner, A. Müller, R. Schuler
On the complexity of intersecting multiple circles for graphical display
2008-02  Manfred Reichert, Peter Dadam, Martin Jurisch, Ulrich Kreher, Kevin Göser, Markus Lauer
Architectural Design of Flexible Process Management Technology

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

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

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

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

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

2008-08  Henning Wunderlich
Covers have structure

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

2008-10  Henning Wunderlich
On span-P^c and related classes in structural communication complexity

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

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

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

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

2009-02  Peter Dadam, Manfred Reichert, Stefanie Rinderle-Ma, Kevin Göser, Ulrich Kreher, Martin Jurisch
2009-03  Alena Hallerbach, Thomas Bauer, Manfred Reichert
Correct Configuration of Process Variants in Provop

2009-04  Martin Bader
On Reversal and Transposition Medians

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

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

Statistical Computing 2009 – Abstracts der 41. Arbeitstagung

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

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

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

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

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

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

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

2010-03  Patrick Frey
Case Study: Engine Control Application

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

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

2010-06  Vera Künzle, Barbara Weber, Manfred Reichert
Object-aware Business Processes: Properties, Requirements, Existing Approaches
2011-01  Stephan Buchwald, Thomas Bauer, Manfred Reichert
Flexibilisierung Service-orientierter Architekturen

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

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

2011-04  Hans A. Kestler, Harald Binder, Matthias Schmid, Friedrich Leisch, Johann M. Kraus (eds):

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

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

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

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

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

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

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

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

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

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

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

2013-02  Andreas Lanz, Manfred Reichert, Barbara Weber
A Formal Semantics of Time Patterns for Process-aware Information Systems
2013-03  Matthias Lohrmann, Manfred Reichert
Demonstrating the Effectiveness of Process Improvement Patterns with Mining Results

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

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

2013-06  Andreas Steigmiller, Birte Glimm, Thorsten Liebig
Nominal Schema Absorption