Optimised Absorption for Expressive Description Logics

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

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

1 Introduction

The Web Ontology Language (OWL 2) [19] is based on the very expressive Description Logic (DL) $\mathcal{SROIQ}$ [3] for which sophisticated algorithms are required to handle (standard) reasoning tasks. In practice, variants of tableau algorithms are often used since they are easily extensible and adaptable. Since such algorithms have a very high worst-case complexity, developing optimisations to nevertheless allow for highly efficient implementations is a long-standing research area in DLs (see, e.g., [4,18]).

A very effective and widely implemented optimisation is *absorption*, which is a preprocessing step that aims at rewriting general concept inclusion (GCI) axioms such that non-determinism in the tableau algorithm is avoided as much as possible. In this paper, we present an improved variant of a recursive binary absorption algorithm that generalises the well-known techniques of binary absorption [7] and role absorption [17]. The algorithm also allows for absorbing parts of concepts and, as a result, more expressive concept constructors can be handled and non-determinism can often be delayed further. The algorithm has already been introduced in our previous work [12] as an essential pre-requisite for handling nominal schemas [8]. Here we simplify its presentation, prove the correctness, present several extensions for the algorithm, and provide a comparison with other absorption techniques. The algorithm is implemented in the tableau-based reasoner Konclude [15] and is essential for Konclude's efficiency. In fact, many other optimisation techniques in Konclude are based on the presented absorption and they significantly benefit from the reduced or eliminated non-determinism [13,14].

We next introduce the basics of tableau algorithms and absorption. In Section 3, we introduce our recursive absorption algorithm for which we present several extension in Section 4. We discuss related work in Section 5 before we conclude in Section 6.

2 Preliminaries

For brevity, we do not introduce DLs (see, e.g., [1,3]) and we only give a short introduction to tableau algorithms (for details see, e.g., [3]). Throughout the paper, we use (possibly with subscripts) C, D for (possibly complex) concepts, A, B for atomic concepts, T for a fresh atomic concept, r for a role, and S for a set of atomic concepts. For simplicity, we also consider $\top$ as an atomic concept. A $\mathcal{SROIQ}$ knowledge base $\mathcal{K}$ is the union of a TBox $\mathcal{T}$ consisting of GCIs of the form $C \sqsubseteq D$, a role hierarchy $\mathcal{R}$, and an ABox $\mathcal{A}$. We write $\mathsf{nnf}(C)$ to denote the negation normal form of C. For r a role, we set $\mathsf{inv}(r) := r^-$ and $\mathsf{inv}(r^-) := r$.

Table 1. Lazy unfolding rules for tableau algorithms

$R\sqsubseteq_1$	if $A \in \mathcal{L}(v), A \sqsubseteq C \in \mathcal{K}, C \notin \mathcal{L}(v)$, and v is not indirectly blocked then $\mathcal{L}(v) = \mathcal{L}(v) \cup \{C\}$
$R\sqsubseteq_2$	if $\{A_1, A_2\} \subseteq \mathcal{L}(v), (A_1 \sqcap A_2) \sqsubseteq C \in \mathcal{K}, C \notin \mathcal{L}(v)$, and v is not indirectly blocked then $\mathcal{L}(v) = \mathcal{L}(v) \cup \{C\}$

Model construction calculi, such as tableau [3,6], 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*. A completion graph G is a tuple $(V, E, \mathcal{L}, \dot{\neq})$, where each node $x \in V$ (edge $\langle x, y\rangle \in E$) represents one or more (pairs of) individuals. Each node x (edge $\langle x, y\rangle$) is labelled with a set of concepts (roles), $\mathcal{L}(x)$ ($\mathcal{L}(\langle x, y\rangle)$), which the (pairs of) individuals represented by x ($\langle x, y\rangle$) are instances of. The relation $\dot{\neq}$ records inequalities, which must hold between nodes, e.g., due to at-least cardinality restrictions. The algorithm works by decomposing concepts in the completion graph with a set of expansion rules. For example, if $C_1 \sqcup C_2 \in \mathcal{L}(v)$ for some node v, but neither $C_1 \in \mathcal{L}(v)$ nor $C_2 \in \mathcal{L}(v)$, then the rule for handling disjunctions non-deterministically adds one of the disjuncts to $\mathcal{L}(v)$. Similarly, if $\exists r.C \in \mathcal{L}(v)$, but v does not have an r-successor with C in its label, then the algorithm expands the completion graph by adding the required successor node. Unrestricted application of this rule or the rule that handles at-least cardinality restrictions can lead to the introduction of infinitely many new tableau nodes. To nevertheless guarantee termination, a cycle detection technique called *(pairwise) blocking* [5] restricts the application of these rules. The rules are repeatedly applied until either the completion graph is fully expanded (no more rules are applicable), in which case the graph can be used to construct a model that *witnesses* the consistency of $\mathcal{K}$, or an obvious contradiction (called a *clash*) is discovered (e.g., both C and $\neg C$ in a node label), proving that the completion graph does not correspond to a model. The input knowledge base $\mathcal{K}$ is *consistent* if the rules can be applied such that they build a fully expanded and clash-free completion graph.

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

$$A \sqsubseteq \exists r.(B_1 \sqcap B_2) \qquad (1) \qquad A \sqcap B_1 \sqcap \exists r.(B_1 \sqcup B_2) \sqsubseteq \exists s.A \qquad (2)$$

the *internalisation* contains one conjunct for each axiom that is a disjunction with the negated left-hand side of the axiom and the right-hand side: $\mathsf{nnf}(\ (\neg A \sqcup \exists r.(B_1 \sqcap B_2)) \sqcap (\neg(A \sqcap B_1 \sqcap \exists r.(B_1 \sqcup B_2)) \sqcup \exists s.A)\ )$. Given that $\top$ is satisfied at each node, a tableau rule can be specified that adds such an internalised concept, say C_I, with an auxiliary axiom of the form $\top \sqsubseteq C_I$ to each node label. 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.

2.1 Absorption

Instead of internalising the TBox axioms, it is 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 also added to the node label. For Axiom (1), for example, it is easy to check whether the atomic concept A is satisfied at a node. This *lazy unfolding* for atomic concepts can be realised by additionally using $\mathrm{R}\sqsubseteq_1$ of Table 1 in the tableau algorithm. With this rule, 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, with Axioms (1) and (2), any instance of A has an r-successor that satisfies $B_1 \sqcap B_2$ and, therefore, this A instance (semantically) also satisfies $\exists r.(B_1 \sqcup B_2)$ (but possibly not B_1). In order to nevertheless avoid the internalisation of such GCIs into axioms of the form $\top \sqsubseteq \mathsf{nnf}(\neg C \sqcup D)$ (and the resulting processing of disjunctions), practical reasoning systems use elaborate transformations in a preprocessing step called *absorption*.

An absorption algorithm rewrites axioms in such a way that internalisation and the processing of disjunctions in the tableau algorithm are avoided as much as possible. To achieve this, the absorption algorithm extracts conditions for an axiom that can easily be checked, and only if the conditions hold for a node in a completion graph, then the remaining (complex) part of the axiom has to be added to the node's label, while otherwise the axiom is trivially satisfied. For example, consider the internalisation $\top \sqsubseteq \neg A \sqcup \neg B_1 \sqcup \forall r.(\neg B_1 \sqcap \neg B_2) \sqcup \exists s.A$ of Axiom (2). We can observe that any node in a tableau that does *not* have an r-neighbour trivially satisfies $\forall r.(\neg B_1 \sqcap \neg B_2)$ and, hence, the overall disjunction. Furthermore, even if there is an r-successor, this is only problematic, if the successor satisfies B_1 or B_2. To check this, we can introduce axioms that make sure that a fresh marker concept T_1 is added to each node label that contains B_1 or B_2, which then triggers the propagation of another marker T_2 to the r-predecessor to indicate that one of the other disjuncts $\neg A$, $\neg B_1$, or $\exists s.A$ has to be satisfied. Since it can also easily be checked whether A or B_1 is satisfied, we can use the axioms:

$$B_1 \sqsubseteq T_1 \qquad B_2 \sqsubseteq T_1 \qquad T_1 \sqsubseteq \forall r^{-}.T_2 \qquad A \sqcap B_1 \sqcap T_2 \sqsubseteq \exists s.A$$

to express Axiom (2) without any disjunction. We say that $\neg A$, $\neg B_1$, and $\forall r.(\neg B_1 \sqcap \neg B_2)$ are *completely absorbable* since they can be moved out of the disjunction, whereas $\exists s.A$ is *non-absorbable*, i.e., it cannot easily be checked when the concept is satisfied and, hence, the concept has to remain on the right-hand side of the axiom.

Note that the last axiom cannot be handled by the lazy unfolding rule $\mathrm{R}\sqsubseteq_1$ since its left-hand side consists of a conjunction of atomic concepts. Instead of adding a rule that processes axioms of the form $A_1 \sqcap \ldots \sqcap A_n \sqsubseteq C$ with A_i atomic concepts, an efficient way of handling such axioms is *binary absorption* [7]. Binary absorption splits such axioms such that A_1 and A_2 imply a fresh concept T_1 ($A_1 \sqcap A_2 \sqsubseteq T_1$), then T_1 is combined with the next condition A_3 and so on, until $T_{n-2} \sqcap A_n \sqsubseteq T_{n-1}$. The concept T_{n-1} then implies the concept C. Hence, rule $\mathrm{R}\sqsubseteq_2$ in Table 1 is sufficient for handling such binary axioms.

3 A Recursive Algorithm for Partial Absorption

We next present an improved variant of a recursive binary absorption algorithm. In comparison to other absorption techniques, the presented approach is often able to further delay non-determinism. The recursion also facilitates further optimisations such as backward chaining for the handling of nominal schemas [12].

Algorithm 1 absorbTBox

Input: $\mathcal{T}$: a TBox, i.e., a set of GCIs $C \sqsubseteq D$
Output: $\mathcal{T}'$: a new TBox with absorbed axioms of $\mathcal{T}$

1: $\mathcal{T}' \leftarrow \emptyset$
2: **for all** $C \sqsubseteq D \in \mathcal{T}$ **do**
3: $\quad S \leftarrow \{A \mid A = \mathsf{absorb}(C', \mathcal{T}'), C' \in \mathsf{PA}(\mathsf{nnf}(\neg C \sqcup D))\}$
4: $\quad A \leftarrow \mathsf{join}(S, \mathcal{T}')$
5: $\quad \{D_1, \ldots, D_m\} \leftarrow \mathsf{notCA}(\mathsf{nnf}(\neg C \sqcup D))$
6: $\quad \mathcal{T}' \leftarrow \mathcal{T}' \cup \{A \sqsubseteq D_1 \sqcup \ldots \sqcup D_m\}$
7: **end for**

Due to the recursion, the presented algorithm works quite differently compared to the original binary absorption [7]. We exploit this for the integration of further improvements. The presented algorithm completely handles an axiom $C \sqsubseteq D$ in one step, i.e., the axiom is first internalised into $\top \sqsubseteq \neg C \sqcup D$ and then the absorbable (sub-)concepts of $\neg C \sqcup D$ are recursively handled. Each recursion step creates (simple) inclusion axioms for the handled (sub-)concepts such that corresponding marker concepts are implied, and the final marker concept (T_{n-1}) is used to imply the non-absorbable part of the axiom. In contrast, the original binary absorption algorithm introduces new GCIs during the absorption of an axiom, which are then (possibly) further absorbed in separate steps.

This complete handling of axioms allows for partially absorbing parts of axioms without creating additional disjunctions. Roughly speaking, this partial absorption also uses parts of non-absorbable concepts as conditions to delay the processing of the non-absorbable part of the axiom. For example, the axiom $\geqslant 5\, r.A \sqsubseteq D$ is, without absorption, handled as $\top \sqsubseteq \leqslant 4\, r.A \sqcup D$. None of the disjuncts can be absorbed completely, but it is nevertheless possible to delay the processing of the disjunction until there is an r-neighbour with the concept A in its label. Partial absorption is able to extract these conditions, and rewrites the axiom such that the disjunction is propagated from a node with A in its label to all r^--neighbours (if there are any), which results in $A \sqsubseteq \forall r^-.T_1$ and $T_1 \sqsubseteq \leqslant 4\, r.A \sqcup D$. Hence, partial absorption can significantly improve the original binary absorption for more expressive Description Logics.

We now make important terms for our partial absorption algorithm clearer.

Definition 1. *A concept C is* completely absorbable *if (i) $C = \neg\{a\}$ for a a nominal, (ii) $C = \neg A$ for A an atomic concept, (iii) $C = C_1 \sqcup \ldots \sqcup C_n$ or $C = C_1 \sqcap \ldots \sqcap C_n$ and, for $1 \leq i \leq n$, C_i is completely absorbable, or (iv) $C = \forall r.D$ and D is completely absorbable. Otherwise C is not completely absorbable. A concept C is* partially absorbable *if C is completely absorbable or (i) $C = \forall r.D$, (ii) $C = \leqslant n\, r.D$, (iii) $C = C_1 \sqcup \ldots \sqcup C_n$ and, for some i, $1 \leq i \leq n$, C_i is partially absorbable, or (iv) $C = C_1 \sqcap \ldots \sqcap C_n$ and, for each i, $1 \leq i \leq n$, C_i is partially absorbable. For a disjunction $C = C_1 \sqcup \ldots \sqcup C_n$, we set*

$$\mathsf{notCA}(C) := \{C_i \mid 1 \leq i \leq n, C_i \text{ is not completely absorbable}\}$$
$$\mathsf{PA}(C) := \{C_i \mid 1 \leq i \leq n, C_i \text{ is partially absorbable}\}.$$

Note that if a concept C is not partially absorbable, then it is also not completely absorbable. In the following, we describe the absorption algorithm.

Algorithm 2 $\mathsf{absorb}(C, \mathcal{T}')$

Input: A partially absorbable concept C and a TBox $\mathcal{T}'$, which is modified via side-effects
Output: Returns the atomic concept for the absorption of C

if $C = \neg A$ **then**
  return A
end if
$T \leftarrow$ fresh atomic concept
if $C = \neg\{a\}$ **then**
  $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{\{a\} \sqsubseteq T\}$
else if $C = \forall r.D$ **then**
  $S \leftarrow \{A \mid A = \mathsf{absorb}(D', \mathcal{T}'), D' \in \mathsf{PA}(D)\}$
  $A \leftarrow \mathsf{join}(S, \mathcal{T}')$
  $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{A \sqsubseteq \forall\, \mathsf{inv}(r).T\}$
else if $C = \leqslant n\, r.D$ **then**
  $S \leftarrow \{A \mid A = \mathsf{absorb}(D', \mathcal{T}'), D' \in \mathsf{PA}(\mathsf{nnf}(\neg D))\}$
  $A \leftarrow \mathsf{join}(S, \mathcal{T}')$
  $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{A \sqsubseteq \forall\, \mathsf{inv}(r).T\}$
else if $C = C_1 \sqcap \ldots \sqcap C_n$ **then**
  $S_C \leftarrow \{\{A \mid A = \mathsf{absorb}(D, \mathcal{T}'), D \in \mathsf{PA}(C_i)\} \mid 1 \leq i \leq n\}$
  $S \leftarrow \{A \mid A = \mathsf{join}(S_i, \mathcal{T}'), S_i \in S_C\}$
  $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{A \sqsubseteq T \mid A \in S\}$
end if
return T

Algorithm 3 $\mathsf{join}(S, \mathcal{T}')$

Input: A set S of atomic concepts and a TBox $\mathcal{T}'$, which is modified via side-effects
Output: Returns an atomic concept that is implied by the join of the concepts in S

while $A_1, A_2 \in S$ **and** $A_1 \neq A_2$ **do**
  $T \leftarrow$ fresh atomic concept
  $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{A_1 \sqcap A_2 \sqsubseteq T\}$
  $S \leftarrow (S \cup \{T\}) \setminus \{A_1, A_2\}$
end while
if $S = \emptyset$ **then return** $\top$
else return the element $A \in S$ ▷ S is a singleton
end if

Algorithm 1 (absorbTBox) takes as input a TBox $\mathcal{T}$ and produces a TBox $\mathcal{T}'$ such that each axiom in $\mathcal{T}'$ has the form $A \sqsubseteq C$, $A_1 \sqcap A_2 \sqsubseteq C$, $\top \sqsubseteq C$, or $\{a\} \sqsubseteq C$, i.e., $\mathcal{T}'$ can be efficiently handled by the rules of Table 1 or as concept assertion. The method absorbTBox considers each axiom $C \sqsubseteq D \in \mathcal{T}$ as disjunction in negation normal form. Each partially absorbable disjunct from $\mathsf{PA}(\mathsf{nnf}(\neg C \sqcup D))$ is passed to the function absorb (Algorithm 2), which (recursively) performs the (binary) absorption and returns a single atomic concept that represents the disjunct. The concepts are then joined and used to imply the non-absorbable part of the axiom in Line 6. Please note that if all disjuncts of the axiom can be completely absorbed, then an empty disjunction is created in Line 6, which we consider as $\bot$.

The method absorb is recursive with the base case of negated concept names ($C = \neg A$) or nominals ($C = \neg\{a\}$). For the former, the algorithm directly returns the atomic concept A. For the latter, the algorithm returns the fresh concept T, which is axiomatised by $\{a\} \sqsubseteq T$ in the TBox constructed during the absorption. For the recursion, partially absorbable subconcepts of C are first absorbed, which results in an atomic concept for each such subconcept. The resulting atomic concepts are then joined by Algorithm 3 (join), which implements the binary absorption introduced in Section 3 and which returns just the final concept (T_{n-1}).

Partial absorption also significantly delays non-determinism for axioms that use more expressive concept constructors. Consider, for example, a TBox $\mathcal{T}$ with

$$\exists r.(A \sqcap \forall r.B_1) \sqsubseteq B_2 \tag{3}$$

as the one and only axiom of the form $C \sqsubseteq D$. Without absorption, it would be necessary to internalise this axiom into the form $\top \sqsubseteq \mathsf{nnf}(\neg C \sqcup D)$, which would result in $\top \sqsubseteq \forall r.(\neg A \sqcup \exists r.\neg B_1) \sqcup B_2$. Obviously, this would produce a significant amount of non-determinism. Clearly, none of the parts of (3) can be absorbed completely, i.e., $\mathsf{notCA}(\mathsf{nnf}(\neg C \sqcup D)) = \{\forall r.(\neg A \sqcup \exists r.\neg B_1), B_2\}$. The concept $\forall r.(\neg A \sqcup \exists r.\neg B_1)$, however, is partially absorbable, i.e., it is in $\mathsf{PA}(\mathsf{nnf}(\neg C \sqcup D))$. This allows for delaying the processing of the disjunctions until there is an r-neighbour with the concept A in its label. In order to capture this, the presented absorption algorithm rewrites the axiom such that the disjunction is propagated from a node with A in its label to all r^--neighbours as follows: In Line 3 of absorbTBox, we call $\mathsf{absorb}(\forall r.(\neg A \sqcup \exists r.\neg B_1), \mathcal{T}')$ with $\mathcal{T}' = \emptyset$. The algorithm proceeds with Lines 7–10 and recursively calls itself for $\neg A$ and $\mathcal{T}'$, which returns A, while $\exists r.\neg B_1$ is not absorbable. The following call of $\mathsf{join}(\{A\}, \mathcal{T}')$ directly returns A (no binarisation is needed). Finally, in Line 10, $\mathcal{T}'$ is extended with $A \sqsubseteq \forall r^-.T$ for the fresh concept T and T is returned by the algorithm. Back in Algorithm 1, the call of $\mathsf{join}(\{T\}, \mathcal{T}')$ again directly returns T. After extending $\mathcal{T}'$ in Line 6 of Algorithm 1, we have $\mathcal{T}' = \{A \sqsubseteq \forall r^-.T, T \sqsubseteq \forall r.(\neg A \sqcup \exists r.\neg B_1) \sqcup B_2\}$.

It is clear that an existential restriction of the form $\exists r.C$ cannot be absorbed directly since it does not provide conditions that could be used to trigger other concepts. In contrast, the absorption can be extended to several concept constructors of more expressive DLs (see also Section 4). For example, the $\neg\exists r.\mathsf{Self}$ concept of $\mathcal{SROIQ}$ can be partially absorbed with $\top \sqsubseteq \forall r^-.A$ and $A \sqsubseteq \neg\exists r.\mathsf{Self}$.

Note that, in principle, it is not necessary to always create new axioms with fresh atomic concepts for the absorption of identical concepts. In practice, the binary absorption axioms as well as the axioms for absorbing specific concepts can be reused.

Lemma 1. *Let $\mathcal{T}$ be a TBox and $\mathcal{T}'$ the TBox produced by Algorithm 1 when given $\mathcal{T}$ as input, then $\mathcal{T}'$ is a model-conservative extension of $\mathcal{T}$, i.e., (a) every model of $\mathcal{T}$ can be extended to a model of $\mathcal{T}'$ by adding suitable interpretations for fresh symbols in $\mathcal{T}'$ and (b) every model of $\mathcal{T}'$ is a model of $\mathcal{T}$ ($\mathcal{T}' \models \mathcal{T}$).*

A complete proof is available in an online technical report [16].

Proof (sketch). Note that by showing $\mathcal{T}' \models \mathcal{T}$, we do not need to require $\mathcal{T} \subseteq \mathcal{T}'$ as usual for (model) conservative extensions.

Let $\mathcal{I}$ be some model of $\mathcal{T}$, $C \sqsubseteq D \in \mathcal{T}$, E a completely (partially) absorbable disjunction that is an absorbed subconcept of $\mathsf{nnf}(\neg C \sqcup D)$, and T_E the concept returned

by $\mathsf{absorb}(E, \mathcal{T}')$. For (a), we construct an interpretation $\mathcal{J}$ for $\mathcal{T}'$ that coincides with $\mathcal{I}$ apart from the interpretations of concepts that are fresh in $\mathcal{T}'$. To show $\mathcal{J} \models \mathcal{T}'$, we establish (1) $(\neg T_E)^{\mathcal{J}} = E^{\mathcal{J}}$ ($(\neg T_E)^{\mathcal{J}} \subseteq E^{\mathcal{J}}$). For showing (b), we first show (2) $\mathcal{T}' \models \neg T_E \sqsubseteq E$.

The base case, where each disjunct of a subconcept $E = E_1 \sqcup \ldots \sqcup E_n$ of $\mathsf{nnf}(\neg C \sqcup D)$ is either non-absorbable or of the form $\neg A$ or $\neg\{a\}$, is straightforward. For the induction, E might also contain other absorbable disjuncts. We just sketch the case of $E_i = \leqslant n\, r.E'$, $1 \leq i \leq n$, where $\mathsf{absorb}(E_i, \mathcal{T}')$ returns T_{E_i} and $T_{\neg E'} \sqsubseteq \forall \mathsf{inv}(r).T_{E_i} \in \mathcal{T}'$, which is equivalent to $\exists r.T_{\neg E'} \sqsubseteq T_{E_i}$. We set $T_{E_i}^{\mathcal{J}} = (\exists r.T_{\neg E'})^{\mathcal{J}}$, so $\mathcal{J} \models T_{\neg E'} \sqsubseteq \forall \mathsf{inv}(r).T_{E_i}$. To show (1): we have $(\neg T_{E_i})^{\mathcal{J}} = (\forall r.(\neg T_{\neg E'}))^{\mathcal{J}}$, and, by induction, $(\forall r.(\neg T_{\neg E'}))^{\mathcal{J}} \subseteq (\leqslant n\, r.E')^{\mathcal{J}} = E_i^{\mathcal{J}}$. For (2): since $\mathcal{T}' \models \exists r.T_{\neg E'} \sqsubseteq T_{E_i}$, $\mathcal{T}' \models \neg T_{E_i} \sqsubseteq \forall r.(\neg T_{\neg E'})$. By induction, $\mathcal{T}' \models \neg T_{\neg E'} \sqsubseteq E'$, hence, $\mathcal{T}' \models \neg T_{E_i} \sqsubseteq \leqslant n\, r.E' = E_i$ as required.

In $\mathsf{absorbTBox}$, we eventually have a concept T_E for the disjuncts $E_1, \ldots, E_n \in \mathsf{PA}(\mathsf{nnf}(\neg C \sqcup D))$ and $T_E \sqsubseteq D_1 \sqcup \ldots \sqcup D_m \in \mathcal{T}'$. By induction, $\mathcal{T}' \models T_{E_1} \sqcap \ldots \sqcap T_{E_n} \sqsubseteq T_E$ and $\mathcal{T}' \models \neg T_{E_i} \sqsubseteq E_i$ for $1 \leq i \leq n$. Hence, $\mathcal{T}' \models \top \sqsubseteq E_1 \sqcup \ldots \sqcup E_n \sqcup D_1 \sqcup \ldots \sqcup D_m$ and, hence, $\mathcal{T}' \models \top \sqsubseteq \mathsf{nnf}(\neg C \sqcup D)$. To show that $\mathcal{J} \models T_E \sqsubseteq D_1 \sqcup \ldots \sqcup D_m$ we show $\mathcal{J} \models \top \sqsubseteq \neg T_{E_1} \sqcup \ldots \sqcup \neg T_{E_n} \sqcup D_1 \sqcup \ldots \sqcup D_m$ since $\mathcal{J} \models T_{E_1} \sqcap \ldots \sqcap T_{E_n} \sqsubseteq T_E$. W.l.o.g., let $E_1, \ldots, E_\ell$ be the completely and $E_{\ell+1}, \ldots, E_n$ the partially, but not completely absorbable disjuncts of E. By induction, $(\neg T_{E_i})^{\mathcal{J}} = E_i^{\mathcal{J}}$ ($(\neg T_{E_i})^{\mathcal{J}} \subseteq E_i^{\mathcal{J}}$), for each $1 \leq i \leq \ell$ ($\ell < i \leq n$). Since $\mathcal{I} \models \mathcal{T}$, $\mathcal{I} \models \top \sqsubseteq E_1 \sqcup \ldots \sqcup E_\ell \sqcup D_1 \sqcup \ldots \sqcup D_m$ and, hence, $\mathcal{J} \models \top \sqsubseteq \neg T_{E_1} \sqcup \ldots \sqcup \neg T_{E_\ell} \sqcup D_1 \sqcup \ldots \sqcup D_m$. Since, for $\ell < i \leq n$, $E_i = D_j$ for some $1 \leq j \leq m$, $\mathcal{J} \models \top \sqsubseteq \neg T_{E_1} \sqcup \ldots \sqcup \neg T_{E_n} \sqcup D_1 \sqcup \ldots \sqcup D_m$ as required.

4 Extensions to the Absorption Algorithm

In this section, we sketch several extensions of the above presented absorption algorithm, which improve the handling of completely defined concepts, optimise the processing of disjunctions, and allow for absorbing axioms that contain data ranges.

4.1 Handling of Completely Defined Concepts

In the previous sections, we considered a TBox to be a set of GCIs of the form $C \sqsubseteq D$. This is w.l.o.g. since $C \equiv D$ can be rewritten to $C \sqsubseteq D$ and $D \sqsubseteq C$. For definitions of the form $A \equiv C$ with A an atomic concept (also called a completely defined concept), it is often inefficient to decompose the axiom $A \equiv C$ into $A \sqsubseteq C$ and $C \sqsubseteq A$, because $\mathsf{nnf}(\neg C)$ might not be completely absorbable and then the disjunction $\neg C \sqcup A$ has to be processed for all nodes in the completion graph. In order to determine the satisfiability of a concept, it is 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 and A is acyclic,[3] then C or $\neg C$ and, therefore, also A or $\neg A$ must be satisfiable and, only in this case,

[3] Acyclicity is defined as follows: A_1 *directly uses* A_2 w.r.t. a TBox $\mathcal{T}$ if $A_1 \equiv C \in \mathcal{T}$ or $A_1 \sqsubseteq C \in \mathcal{T}$ 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* $\mathcal{T}$ if it contains no concept A that uses itself.

1: **for all** $A \equiv C \in \mathcal{T}$ **do**
2: $T_A \leftarrow \mathsf{absorb}(\mathsf{PA}(\mathsf{nnf}(\neg C)), \mathcal{T}')$
3: **if** $\mathsf{nnf}(\neg C)$ is completely absorbable **then**
4: $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{A \sqsubseteq C, T_A \sqsubseteq A, T_A \sqsubseteq A^+\}$
5: **else if** $|\{C' \mid A \sqsubseteq C' \in \mathcal{T} \text{ or } A \equiv C' \in \mathcal{T}\}| > 1$ **then**
6: $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{A \sqsubseteq C, T_A \sqsubseteq \mathsf{nnf}(\neg C \sqcup A), T_A \sqsubseteq A^+\}$
7: **else**
8: $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{A \equiv C, T_A \sqsubseteq A^+\}$
9: **end if**
10: **end for**

Fig. 1. Extension for Algorithm 1 for completely defined concepts (to be inserted after Line 1)

the axiom $A \equiv C$ can directly be handled by an additional rule, which unfolds A to C and $\neg A$ to $\neg C$. To capture this, we extend the algorithm absorbTBox (cf. Figure 1) such that axioms of the from $A \equiv C$ are only eliminated if there are several definitions of A. In such cases, $A \sqsubseteq C$ and $C \sqsubseteq A$ must be explicitly represented in $\mathcal{T}'$ such that possible interactions between these several definitions are handled. In addition, a concept of the form $\neg A$ is now only *completely (partially) absorbable* if A is acyclic and, for each $A \equiv C \in \mathcal{T}$, $\mathsf{nnf}(\neg C)$ is completely (partially) absorbable. Note that the definition of completely and partially absorbable concepts now refers to the TBox. To correctly handle the completely defined concepts in the recursive absorption algorithm, absorb also has to be adapted for concepts of the form $\neg A$. The algorithm still returns A if there is no axiom of the form $A \equiv C \in \mathcal{T}$. Otherwise, for each $A \equiv C$, all absorbable disjuncts of $\mathsf{nnf}(\neg C)$ are recursively absorbed, the resulting atomic concepts are joined into a fresh concept T_C, and $T_C \sqsubseteq A^+$ is added to $\mathcal{T}'$ for a so-called *candidate concept* A^+ for A. After processing all definitions for A, the candidate concept A^+ is returned. Note that if $\mathsf{nnf}(\neg C)$ is not absorbable, then the absorption returns $\top$ and $\top \sqsubseteq A^+$ is added to $\mathcal{T}'$. An occurrence of A^+ in the label of a node signalises that the node might be an instance of the concept A, i.e., if A^+ is not in the node label of a clash-free and fully expanded completion graph, then this node is also not an instance of A. In particular, if A^+ is not in the label of a node, then we know that $\neg A$ can be safely added since one disjunct of $\mathsf{nnf}(\neg C)$ is trivially satisfied. Hence, the generated candidate concept A^+ can be used as condition for further absorption if $\neg A$ occurs and, therefore, it can be used to further delay branching.

Such candidate concepts are also very useful for identifying completely defined concepts as possible subsumers for classification (cf. [2]) without forcing the decision between A and $\neg A$ for the nodes in the completion graph. As a result, these candidate concepts can significantly improve the classification performance, especially in combination with model merging techniques [13,14], and, therefore, it is almost always useful to generate these candidate concepts even if they are not required for further absorption.

4.2 Handling of Disjunctions

The presented absorption algorithm can also be used to optimise the handling of ordinary disjunctions. For example, the axiom $A \sqsubseteq \exists r.(\neg B \sqcup A)$ cannot be absorbed and

contains the disjunction $\neg B \sqcup A$. Obviously, we can replace the disjunction with a fresh marker concept T by adding the additional axiom $T \sqsubseteq \neg B \sqcup A$ to the TBox, which can be absorbed to $T \sqcap B \sqsubseteq A$. Consequently, we have the axioms $A \sqsubseteq \exists r.T$ and $T \sqcap B \sqsubseteq A$, which can be processed by the tableau algorithm without causing any non-determinism.

4.3 Handling of Datatypes

Another advantage of the presented absorption algorithm is the straightforward extensibility to handle *datatypes* such as strings, integers, or decimals, which represent sets of concrete data values. In OWL, one can also use *datatype restrictions* to build custom datatypes, e.g., the datatype restriction $\mathit{real}[\mathit{int} \wedge >_5]$ restricts OWL's datatype *owl*:*real*, which we abbreviate to *real*, with the *facets int* and $>_5$ to real numbers that are integers and greater than 5. We call $\mathit{int} \wedge >_5$ a *facet expression*. One can further use Boolean constructors to build *data ranges*, e.g., $\mathit{string} \vee \mathit{real}[\mathit{int} \wedge >_5]$ is a data range that contains all data values that are strings or reals as described above. Concepts refer to data ranges via *concrete roles*, e.g., the formula $\exists q.\mathit{real}[\mathit{int} \wedge >_5]$, for q a concrete role, describes individuals that have some value of the data range $\mathit{real}[\mathit{int} \wedge >_5]$ as q-successor.

Elements that represent data values are represented in the tableau algorithm by *concrete nodes*. For example, for a node v with $\exists q.\mathit{real}[\mathit{int} \wedge >_5] \in \mathcal{L}(v)$, the tableau algorithm adds a new *concrete node* z and an edge $\langle v, z\rangle$ with $q \in \mathcal{L}(\langle v, z\rangle)$ and $\mathit{real}[\mathit{int} \wedge >_5] \in \mathcal{L}(z)$. Typically, the tableau algorithm is then combined with a datatype checker that checks the consistency of datatype constraints [10]. Such constraints are represented as a set of assertions of the form $d(z)$, $\neg d(z)$, and $z_1 \not\approx z_2$, where z, z_1, and z_2 are *concrete nodes* and d is an explicit *enumeration* of data values $\{c_1, \ldots, c_n\}$ or an expression of the form $dt[\varphi]$ for dt a datatype and φ a *facet expression*. Given a set of such assertions, the datatype checker determines whether each concrete node can be assigned a data value in a way that satisfies all of the given assertions.

It is well-known that facets can be used to encode disjunctions [9]. For example, $A \sqsubseteq \forall r.B_1 \sqcup B_2$ can be expressed by the axioms

$$A \sqsubseteq \exists q.\mathit{real}[\mathit{int} \wedge >_5] \quad (4) \qquad \exists q.\mathit{real}[\leq_{10}] \sqsubseteq \forall r.B_1 \quad (5) \qquad \exists q.\mathit{real}[>_{10}] \sqsubseteq B_2, \quad (6)$$

where q is a concrete role. For the data range $\mathit{real}[\mathit{int} \wedge >_5]$, it cannot directly be determined whether $\mathit{real}[\leq_{10}]$ or its complement is satisfied. This has to be clarified in a case-by-case analysis, where once the integer interval $\mathit{int}(5, 10]$ and once the interval $\mathit{int}(10, \infty]$ is considered. Hence, the datatype checker is forced to make nondeterministic decisions. This also means that it is usually not possible to (completely) absorb data ranges since it is not easily possible to determine if a certain data range is satisfied or not. Of course, it is possible to absorb other parts of the axioms, e.g., Axiom (5) can be absorbed to

$$\top \sqsubseteq \forall r^-.T \qquad\qquad T \sqsubseteq \forall q.\overline{\mathit{real}[\leq_{10}]} \sqcup \forall r.B_1,$$

where $\overline{\mathit{real}[\leq_{10}]}$ expresses the negation of $\mathit{real}[\leq_{10}]$. However, without any further optimisations, Axiom (6) introduces non-determinism for every node as it is handled as

$$\top \sqsubseteq \forall q.\overline{\mathit{real}[>_{10}]} \sqcup B_2.$$

For partial absorption it is only required that the tableau algorithm or the datatype checker can identify those concepts and data ranges that might be satisfied. This is obviously possible for many data ranges and, therefore, they can be absorbed partially.

For this, the datatype checker has to be extended such that it identifies data range candidates, i.e., data ranges that are possibly satisfied. Let us assume that $real[\leq_{10}]^+$ is the data range candidate for $real[\leq_{10}]$, i.e., if a concrete node in the completion graph possibly represents numerical data values that are less or equal to 10, then $real[\leq_{10}]^+$ is satisfied and this can be identified efficiently by the datatype checker. Consequently, we can use such data range candidates in lazy unfolding rules, e.g., $real[\leq_{10}]^+ \sqsubseteq T$ is used to trigger the marker concept T if $real[\leq_{10}]$ is possibly satisfied. Note that this adds ordinary concepts to the labels of concrete nodes, which is, however, not problematic since these concepts are only used internally as auxiliary constructs. With the data range candidates, we can improve the absorption of Axiom (5) to

$$real[\leq_{10}]^+ \sqsubseteq T_1 \quad T_1 \sqsubseteq \forall q^-.T_2 \quad \top \sqsubseteq \forall r^-.T_3 \quad T_2 \sqcap T_3 \sqsubseteq \forall q.\overline{real[\leq_{10}]} \sqcup \forall r.B_1$$

and we can absorb Axiom (6) to

$$real[>_{10}]^+ \sqsubseteq T_4 \qquad T_4 \sqsubseteq \forall q^-.T_5 \qquad T_5 \sqsubseteq \forall q.\overline{real[>_{10}]} \sqcup B_2.$$

Now, the processing of the disjunctions is delayed until there are concrete nodes that possibly represent the absorbed data ranges.

The required extensions for the absorption algorithm and the datatype checker are straightforward. For the absorption, the definition of partial absorbability has to be extended to the supported data ranges and the concepts related to datatypes. Moreover, the absorb function has to be adjusted such that the corresponding axioms are created. The datatype checker has to be extended to identify those data range candidates that might be satisfied. Usually, the datatype checker manages the data intervals that are still possible for a concrete node in a sorted array [10]. By counting and iterating through the possible values, it is easily possible to identify those data range candidates that are satisfied. For example, the possible values of a concrete node z are restricted to the integer interval $int(5, \infty]$ if the data range $real[int \wedge >_5]$ is asserted to z. If z is not distinct to other concrete nodes (i.e., there is no assertion $z \not\approx z'$), then the datatype checker can pick the first possible data value of 6 to satisfy the constraints for z. Hence, the datatype checker has to trigger only those data range candidates that are satisfied for this first data value (i.e., $real[\leq_{10}]^+$). If z is distinct to 5 other concrete nodes, then $real[>_{10}]^+$ is also triggered (although it is not necessarily the case that indeed 6 data values are required to assign all (distinct) concrete nodes a data value).

5 Comparison with Related Absorption Techniques

In comparison to other absorption techniques, *partial absorption* often delays nondeterminism further. Consider again Axiom (3) from Section 3, which is rewritten by the original binary absorption algorithm into the following axioms:[4]

$$A \sqsubseteq \exists r.\neg B_1 \sqcup \forall r^-.T \qquad T \sqsubseteq B_2.$$

In contrast, partial absorption creates the following axioms (as shown in Section 3):

$$A \sqsubseteq \forall r^-.T \qquad T \sqsubseteq \forall r.(\neg A \sqcup \exists r.\neg B_1) \sqcup B_2.$$

[4] The binary absorption algorithm presented in [7] has a looping bug for concepts of the form $\forall r.C$, which can, however, easily be repaired. We simply ignored the repeated absorption of concepts that are introduced by the algorithm.

Hence, with the original binary absorption technique, the disjunction has already to be processed when A is added to the label of a node, whereas partial absorption delays the processing until there is also an r-neighbour. It is also well-known that clausification for the hypertableau algorithm reduces many cases of non-determinism [11]. For Axiom (3), however, clausification creates the following DL-clauses:

$$r(x, y) \rightarrow B_2(x) \vee T(y) \qquad\qquad T(x) \wedge A(x) \rightarrow \exists r.\neg B_1(x)$$

Hence, the hypertableau algorithm has to choose between B_2 and T for every node with an r-neighbour, whereas the partial absorption technique allows for further delaying the processing of disjunctions until the node also has the concept A in its label. Besides that, the presented partial absorption technique shares other interesting features with clausification of the hypertableau algorithm. In particular, both use conditions of more expressive concept constructors to trigger the processing of disjunctions. For instance, the axiom $\geqslant 2\, r.A \sqsubseteq B$ is automatically satisfied if there is no r-successor with the concept A in its label. This is utilised by partial absorption, which generates

$$A \sqsubseteq \forall r^{-}.T \qquad\qquad T \sqsubseteq \leqslant 1\, r.A \sqcup B$$

and also by clausification of the hypertableau calculus, which generates:

$$r(x, y_1) \wedge A(y_1) \wedge r(x, y_2) \wedge A(y_2) \rightarrow B(x) \vee y_1 \approx y_2$$

Note that clausification eliminates at-most cardinality restrictions, whereas partial absorption has to use the concept $\leqslant 1\, r.A$ since $\geqslant 2\, r.A$ is not completely absorbable. Consequently, a choose-rule is not required in the hypertableau algorithm, which possibly further reduces non-determinism in comparison to a standard tableau algorithm that has to choose between $\neg A$ and A for all r-neighbours of nodes with T in their label.

By using a *recursive* algorithm that absorbs GCIs completely in one step, we gain a further advantage in comparison to traditional absorption algorithms. Traditional algorithms absorb parts of axioms in separate steps, e.g., by introducing new GCIs that are handled separately, which often creates additional disjunctions, especially if the axioms have more complex (sub-)concepts. For instance, by simplifying the axiom $\geqslant 5\, r.(\geqslant 3\, s.A) \sqsubseteq D$ into $\geqslant 5\, r.T \sqsubseteq D$ and $T \equiv \geqslant 3\, s.A$, for T a fresh atomic concept that is used to split the original axiom, the GCI $T \equiv \geqslant 3\, s.A$ cannot further be absorbed without creating additional disjunctions. In contrast, our recursive absorption algorithm creates the axioms $A \sqsubseteq \forall s^{-}.T_1$, $T_1 \sqsubseteq \forall r^{-}.T_2$, and $T_2 \sqsubseteq \leqslant 4\, r.(\geqslant 3\, s.A) \sqcup D$, which delays the processing of the disjunction significantly (or avoids it completely if there is no node in the completion graph for which T_2 is added to its label).

6 Conclusions

In this paper, we have presented the partial absorption algorithm used in the reasoning system Konclude together with several extensions. The candidate concepts can, for example, be combined with model merging techniques to reduce Konclude's classification time for several thousand ontologies by 60 % on average [13,14], which allows Konclude to often outperform other state-of-the-art reasoners. The presented absorption techniques are also essential for further optimisations such as an efficient handling of nominal schemas [12] or the combination of tableau with saturation-based procedures [13,14], where the reduced non-determinism is crucial.

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, second edn. (2007)
2. Glimm, B., Horrocks, I., Motik, B., Shearer, R., Stoilos, G.: A novel approach to ontology classification. J. of Web Semantics: Science, Services and Agents on the World Wide Web 14, 84–101 (July 2012)
3. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: Proc. 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'06). pp. 57–67. AAAI Press (2006)
4. Horrocks, I., Patel-Schneider, P.F.: Optimizing description logic subsumption. J. of Logic and Computation 9(3), 267–293 (1999)
5. Horrocks, I., Sattler, U.: A description logic with transitive and inverse roles and role hierarchies. J. of of Logic and Computation 9(3), 385–410 (1999)
6. Horrocks, I., Sattler, U.: A tableau decision procedure for $\mathcal{SHOIQ}$. J. of Automated Reasoning 39(3), 249–276 (2007)
7. Hudek, A.K., Weddell, G.E.: Binary absorption in tableaux-based reasoning for description logics. In: Proc. 19th Int. Workshop on Description Logics (DL'06). vol. 189. CEUR (2006)
8. Krötzsch, M., Maier, F., Krisnadhi, A., Hitzler, P.: A better uncle for OWL: nominal schemas for integrating rules and ontologies. In: Proc. 20th Int. Conf. on World Wide Web (WWW'11). pp. 645–654. ACM (2011)
9. Magka, D., Kazakov, Y., Horrocks, I.: Tractable extensions of the description logic $\mathcal{EL}$ with numerical datatypes. J. of Automated Reasoning 47(4), 427–450 (2011)
10. Motik, B., Horrocks, I.: OWL datatypes: Design and implementation. In: Proc. 7th Int. Semantic Web Conf. (ISWC'08). LNCS, vol. 5318, pp. 307–322. Springer (2008)
11. Motik, B., Shearer, R., Horrocks, I.: Hypertableau reasoning for description logics. J. of Artificial Intelligence Research 36, 165–228 (2009)
12. Steigmiller, A., Glimm, B., Liebig, T.: Nominal schema absorption. In: Proc. 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI'13). pp. 1104–1110. AAAI Press/The MIT Press (2013)
13. Steigmiller, A., Glimm, B., Liebig, T.: Coupling tableau algorithms for expressive description logics with completion-based saturation procedures. In: Proc. 7th Int. Joint Conf. on Automated Reasoning (IJCAR'14). LNCS, Springer (2014), accepted
14. Steigmiller, A., Glimm, B., Liebig, T.: Coupling tableau algorithms for the DL $\mathcal{SROIQ}$ with completion-based saturation procedures. Tech. Rep. UIB-2014-02, University of Ulm, Ulm, Germany (2014), available online at `http://www.uni-ulm.de/fileadmin/website_uni_ulm/iui/Ulmer_Informatik_Berichte/2014/UIB-2014-02.pdf`
15. Steigmiller, A., Liebig, T., Glimm, B.: Konclude: system description. J. of Web Semantics (2014), accepted
16. Steigmiller, A., Liebig, T., Glimm, B.: Optimised absorption for expressive description logics. Tech. rep., Ulm University, Ulm, Germany (2014), available online at `https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.090/Publikationen/2014/StGL14c.pdf`
17. Tsarkov, D., Horrocks, I.: Efficient reasoning with range and domain constraints. In: Proc. 17th Int. Workshop on Description Logics (DL'04). vol. 104. CEUR (2004)
18. Tsarkov, D., Horrocks, I., Patel-Schneider, P.F.: Optimizing terminological reasoning for expressive description logics. J. of Automated Reasoning 39, 277–316 (2007)
19. W3C OWL Working Group: OWL 2 Web Ontology Language: Document Overview. W3C Recommendation (27 October 2009), available at `http://www.w3.org/TR/owl2-overview/`