Absorption-Based Query Entailment Checking for Expressive Description Logics

Andreas Steigmiller[*] and Birte Glimm

Ulm University, Ulm, Germany, <first name>.<last name>@uni-ulm.de

Abstract. Conjunctive query answering is an important reasoning task for logic-based knowledge representation formalisms, such as Description Logics, to query for instance data that is related in certain ways. Although there exist many knowledge bases that use language features of more expressive Description Logics, there are hardly any systems that support full conjunctive query answering for these logics. In fact, existing systems usually impose restrictions on the queries or only compute incomplete results.

In this paper, we present a new approach for conjunctive query entailment checking that can directly be integrated into existing reasoning systems for expressive Description Logics and serves as basis for conjunctive query answering. The approach reminds of *absorption*, a well-known preprocessing step that rewrites axioms such that they can be handled more efficiently. In this sense, we rewrite the query such that entailment can dynamically be checked in the dominantly used tableau calculi with minor extensions. Our implementation in the reasoning system Konclude shows encouraging results.

1 Introduction

Although conjunctive query answering has intensively been studied for many expressive Description Logics (DL), most of the state-of-the-art reasoning systems for these DLs do not support conjunctive queries or only with limitations. In fact, conjunctive query answering is typically reduced to many query entailment checking problems for which decidability is still open in $\mathcal{SROIQ}$. Furthermore, the used techniques for showing decidability and worst-case complexity of sub-languages (e.g., [3,13,16]) are often not directly suitable for practical implementations. For the DLs $\mathcal{SHIQ}$ and $\mathcal{SHOQ}$ approaches have been developed that reduce conjunctive query answering to instance checking (e.g, [4,6,10]), which is not goal-directed, often requires many unnecessary entailment checks, and may require language features (e.g., role conjunctions) which are not available in OWL 2 and, hence, usually not supported by reasoning systems. Even for queries with only answer variables, existing approaches (e.g., [8,12,17]) are often suboptimal since they are based on the above mentioned reduction to many instance checks. Recently, query answering has been improved by lower and upper bound optimisations that utilise model abstractions built by a reasoner [5] or delegate work to specialised procedures [14,22]. However, their effectiveness is ontology dependent and, hence, optimised and deeply integrated query entailment checking and answering techniques for expressive DLs are still needed for further improvements.

[*] Funded by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG), project number 330492673

In this paper, we present an approach that encodes the query such that entailment can efficiently be detected in the model construction process with minor extensions to the dominantly used tableau algorithm. The encoding serves to identify individuals involved in satisfying the query and guides the search for a model where the query is not entailed. We refer to this technique as *absorption-based query entailment checking* since it reminds of the absorption technique for nominal schemas [19]. The approach is correct and terminates for several expressive DLs for which decidability of conjunctive query entailment is well-known (e.g., $\mathcal{SHIQ}$, $\mathcal{SHOQ}$). For the challenging combination of nominals, inverse roles, and number restrictions, termination is only guaranteed if a bounded number of new nominals is generated. The technique seems well-suited for practical implementations since (i) it only requires minor extensions to tableau algorithms, (ii) can easily be combined with other well-known (query answering) optimisation techniques, and (iii) real-world ontologies hardly require the generation of (many) new nominals. We implemented the proposed technique in the reasoning system Konclude [20] with encouraging results. The accompanying technical report [18] sketches extensions to query answering and contains more details as well as evaluation results.

2 Preliminaries

We assume readers to be familiar with DLs and conjunctive queries (see, e.g., [1]). Since we focus on query entailment checking, we only consider Boolean queries (i.e., all variables are existential variables). A query Q is a set of query terms $\{q_1, \ldots, q_k\}$ with q_i either a concept term of the form $C(z)$ or a role term of the form $r(z_1, z_2)$. We consider $r(x, y) \in Q$ as equivalent to $r^-(y, x) \in Q$ and use $\mathsf{vars}(Q)$ to refer to the variables in Q. W.l.o.g. we do not use individual names in query terms and we assume that all variables are connected via role terms.

Reasoning algorithms for $\mathcal{SROIQ}$ are dominantly based on (variants of) tableau algorithms, which, roughly speaking, check 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 $v \in V$ (edge $\langle v, w\rangle \in E$) represents one or more (pairs of) individuals. Each node v (edge $\langle v, w\rangle$) is labelled with a set of concepts (roles), $\mathcal{L}(v)$ ($\mathcal{L}(\langle v, w\rangle)$), which the individuals represented by v ($\langle v, w\rangle$) are instances of. The relation $\dot{\neq}$ records inequalities between nodes. We call $C \in \mathcal{L}(v)$ ($r \in \mathcal{L}(\langle v, w\rangle)$) a *concept (role) fact*, which we write as $C(v)$ ($r(v, w)$). A node v is a *nominal node* if $\{a\} \in \mathcal{L}(v)$ for some individual a and a *blockable node* otherwise.

A completion graph is initialised with one node for each individual in the input knowledge base. Concepts and roles are added to the node and edge labels as specified by concept and role assertions. Complex concepts are then decomposed using expansion rules, where each rule application can add new concepts to node labels and/or new nodes and edges, thereby explicating the structure of a model. The rules are applied until either the graph is fully expanded (no more rules are applicable), in which case the graph can be used to construct a model that is a *witness* to the consistency of $\mathcal{K}$, or an obvious contradiction (called a *clash*) is discovered (e.g., a node v with $C, \neg C \in \mathcal{L}(v)$), proving that the completion graph does not correspond to a model. $\mathcal{K}$ is *consistent* if the rules (some of which are non-deterministic) can be applied such that they build a fully

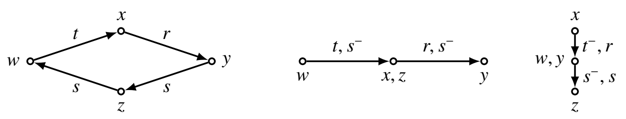


Fig. 1. Visualisation of the query of Example 1 and two possible foldings

expanded, clash-free completion graph. Cycle detection techniques such as *pairwise blocking* [9] prevent the infinite generation of new nodes.

For handling axioms of the form $A \sqsubseteq C$, where A is atomic, one typically uses special *lazy unfolding* rules in the tableau algorithm, which add C to a node label if it contains the concept A. Axioms of the form $C \sqsubseteq D$, where C is not atomic, cannot directly be handled with lazy unfolding rules. Instead, they are internalised to $\top \sqsubseteq \neg C \sqcup D$. Given that $\top$ is satisfied at each node, the disjunction is then present in all node labels. To avoid the non-determinism introduced by internalisation, one typically uses a preprocessing step called *absorption* to rewrite axioms into (possibly several) simpler concept inclusion axioms that can be handled by lazy unfolding. Binary absorption [11] utilises axioms of the form $A_1 \sqcap A_2 \sqsubseteq C$ for absorbing more complex axioms. This requires a binary unfolding rule that adds C to node labels if A_1 and A_2 are present.

3 Absorption-Based Query Entailment Checking

With the exception of role relationships between nominals/individuals, DLs allow only for expressing tree-shaped structures [7,21]. Even with nominals/individuals, forest-shaped models exists [16]. Hence, we can check query entailment by "folding" the relational structure of (parts of) the query into a tree-shaped form by identifying variables. The resulting queries (query parts), called foldings, can then be expressed as DL concepts (possibly using role conjunctions). Such query concepts can be used to check query entailment: we have that a query (part) is not entailed if a completion graph exists that satisfies none of its foldings.

Example 1. Consider the (cyclic) query $Q_1 = \{t(w, x), r(x, y), s(y, z), s(z, w)\}$ (cf. Figure 1, left-hand side). There are different (tree-shaped) foldings of the query, e.g., by identifying x and z or w and y (cf. Figure 1, middle and right-hand side). The foldings can be expressed as $\exists(t \sqcap s^-).\exists(r \sqcap s^-).\top$ and $\exists(t^- \sqcap r).\exists(s^- \sqcap s).\top$, respectively.

If we add, for each concept, say C, that represents a folding of the query, the axiom $C \sqsubseteq \bot$ to the knowledge base, then consistency checking reveals query entailment. Note that the tableau algorithm decides for each node whether (sub-)concepts of the foldings are satisfied (due to the internalisation to $\top \sqsubseteq \neg C \sqcup \bot$) and adds corresponding (sub-)concepts or their negations to the node labels and, hence, the expansion of nodes is not blocked too early w.r.t. deciding query entailment. Unfortunately, state-of-the-art reasoners do not support role conjunctions and there can be many foldings of a query (especially if the query has several nested cycles or uses role terms with complex roles).

Here we propose, as an alternative, to dynamically match and fold the query onto the completion graph. This is achieved by 'absorbing' a query into several simple axioms that can efficiently be processed, where intermediate states encode the parts of the query that are already satisfied. The intermediate states are tracked in the form of so-called *query state concepts* (written S, possibly with sub-/super-scripts), which can be seen as fresh atomic concepts with a set of associated bindings of query variables to nodes in the completion graph. To realise this, we extend the tableau algorithm to *create variable bindings* (to match a variable to a node in the completion graph), to *propagate variable bindings* in the process of folding the query onto the completion graph, and to *join variable bindings*. Creating and propagating variable bindings according to the role terms of a query ultimately allows us to detect when cycles are closed.

For the creation of variable bindings, we borrow the $\downarrow$ binders from Hybrid Logics [2]. Informally, a concept of the form $\downarrow x.C$ in the label of a node v instructs the tableau algorithm to create a binding $\{x \mapsto v\}$, which binds x to the node v, and to store the binding for the sub-concept C. For the propagation of bindings, we extend the $\forall$-rule of the tableau algorithm. For example, if $\forall r.C$ is in the label of a node v and the variable binding $\{x \mapsto v\}$ is associated with it, then the tableau algorithm associates $\{x \mapsto v\}$ with C for all r-successors of v. Additionally, propagation can happen within node labels, e.g., if $S \in \mathcal{L}(v)$ with the associated binding $\{x \mapsto v\}$ and the knowledge base contains $S \sqsubseteq C$, we add C to $\mathcal{L}(v)$ and associate $\{x \mapsto v\}$ with it. Finally, for joining bindings, we extend the binary unfolding rule. For example, if S_1 and S_2 are in the label of a node v and the bindings $\{x \mapsto v, y \mapsto w\}$ and $\{x \mapsto v, z \mapsto w\}$ are associated with them, respectively, then, for an axiom $S_1 \sqcap S_2 \sqsubseteq C$ in the knowledge base, we add C to $\mathcal{L}(v)$ and associate the joined bindings $\{x \mapsto v, y \mapsto w, z \mapsto w\}$ with it. With these basic adaptations, we can capture the query in several simple types of axioms: $S \sqsubseteq \downarrow x.S'$ for creating bindings, $S \sqsubseteq S'$ and $S \sqsubseteq \forall r.S'$ for propagating bindings, and $S_1 \sqcap S_2 \sqsubseteq S'$ for joining bindings, where S, S', S_1, S_2 are query state concepts and r is an atomic role or its inverse. The resulting axioms can usually be processed quite efficiently.

3.1 Query Absorption

Before presenting a formal algorithm, we demonstrate how the concepts and axioms for a query are obtained by means of an example. We call this process *absorbing a query*.

Example 2 (Example 1 cont.). Consider again $Q_1 = \{t(w,x),\ r(x,y),\ s(y,z),\ s(z,w)\}$. We first pick a starting variable, say w, and introduce the axiom $\top \sqsubseteq \downarrow w.S^w$, which triggers, for all nodes, that a binding for w is created. We use the (fresh) query state concept S^w to indicate that w is bound. Since it is convenient to continue with a role term containing w, we choose $t(w,x)$ and propagate the bindings for w to t-successors using the axiom $S^w \sqsubseteq \forall t.S^w_t$ (again S^w_t is fresh and indicates the state that bindings for w have been propagated via t). Nodes to which S^w_t (with the bindings for w) is propagated are suitable bindings for x. This is captured by the axiom $S^w_t \sqsubseteq \downarrow x.S^x$. Since S^w_t may be propagated from different nodes, we join the propagated bindings for w and the newly created bindings for x using the axiom $S^w_t \sqcap S^x \sqsubseteq S^{wx}$, for which the extended tableau algorithm attaches the joined bindings to the fresh concept S^{wx}. We proceed analogously for $r(x,y)$, $s(y,z)$, and $s(z,w)$ (see Figure 2 for all created axioms). Nodes to which the concept S^{wxyz}_s is propagated, potentially close the cycle in the query. The axiom $S^{wxyz}_s \sqcap S^w \sqsubseteq S^{wxyzw}$ checks whether a join is possible. In case it is, the query is satisfied

$$\begin{array}{rrrr}
\top \sqsubseteq {\downarrow}w.S^{w} & S^{w} \sqsubseteq \forall t.S^{w}_{t} & S^{w}_{t} \sqsubseteq {\downarrow}x.S^{x} & \\
S^{w}_{t} \sqcap S^{x} \sqsubseteq S^{wx} & S^{wx} \sqsubseteq \forall r.S^{wx}_{r} & S^{wx}_{r} \sqsubseteq {\downarrow}y.S^{y} & \\
S^{wx}_{r} \sqcap S^{y} \sqsubseteq S^{wxy} & S^{wxy} \sqsubseteq \forall s.S^{wxy}_{s} & S^{wxy}_{s} \sqsubseteq {\downarrow}z.S^{z} & \\
S^{wxy}_{s} \sqcap S^{z} \sqsubseteq S^{wxyz} & S^{wxyz} \sqsubseteq \forall s.S^{wxyz}_{s} & S^{wxyz}_{s} \sqcap S^{w} \sqsubseteq S^{wxyzw} & S^{wxyzw} \sqsubseteq \bot
\end{array}$$

Fig. 2. The axioms for absorbing the query Q_1 of Example 2

and a clash is triggered by the axiom $S^{wxyzw} \sqsubseteq \bot$. In this case, backtracking is potentially triggered to try other non-deterministic choices which might yield a complete and clash-free completion graph that is a counter example for the query entailment.

The next example demonstrates how concept terms in the query are handled.

Example 3 (Example 2 continued). Consider $Q_2 = Q_1 \cup \{B(x)\}$. As in Example 2, we pick w as starting node and then process $t(w, x)$. This yields (again) the first four axioms shown in Figure 2. Assume that we now process $B(x)$. At the state S^{wx}, the tableau algorithm can either satisfy $\neg B$ (which indicates that the query is not satisfied with these bindings for w and x) or we have to assume a query state where also $B(x)$ is satisfied. This is achieved by adding the axiom $S^{wx} \sqsubseteq \neg B \sqcup F^{x}_{B}$, where F^{x}_{B} is a fresh concept. Note that we want to keep the number of modified tableau rules minimal. Hence, when applied to $\neg B \sqcup F^{x}_{B}$, the $\sqcup$-rule does not propagate variable bindings. In case, the disjunct F^{x}_{B} is chosen, we join its empty set of variable bindings with those for S^{wx} using the axiom $S^{wx} \sqcap F^{x}_{B} \sqsubseteq S^{wx}_{B}$, which is handled by the extended binary unfolding rule. For the next role term $r(x, y)$, we then add $S^{wx}_{B} \sqsubseteq \forall r.S^{wx}_{r}$ and continue as in Example 2.

Algorithm 1 formalizes the query absorption process and extends the given knowledge base $\mathcal{K}$ via side effects. The functions absorbCT (Algorithm 2) and absorbRT (Algorithm 3) handle concept and role terms, respectively. The functions use a mapping V_{LS} from **v**ariables to the **l**ast query **s**tate concepts, i.e., each variable in the query is mapped to the last introduced query state concept for that variable such that we can later continue or incorporate the propagation for that variable. In the example, we always chose an adjacent next query term that contains the current variable z. In case a non-adjacent term is chosen, Lines 6–11 ensure the connection to the current variable (which exists as we consider connected queries, see Section 2). In our example, if we were to choose $s(y, z)$ as first query term in Line 5 (with w as starting variable), Lines 6–11 ensure that we process, for example, $t(w, x)$ and $r(x, y)$ before we process $s(y, z)$ in Line 17. Clearly, the presented algorithm can further be optimised, e.g., by not creating binder concepts for variables that are not required in joins, but the presented algorithm is already quite convenient to show the principle of the approach.

3.2 Tableau Rules and Blocking Extensions

As outlined in the previous sections, minor extensions and adaptations of the tableau algorithm are required for creating, propagating, and joining bindings as well as for

Algorithm 1 absorbQ($Q, \mathcal{K}$)

```
Input: A query $Q$ and a knowledge base $\mathcal{K}$
       that is extended via side effects
 1: $z \leftarrow$ choose one variable from $\mathsf{vars}(Q)$
 2: $S^z \leftarrow$ fresh query state concept
 3: $\mathcal{K} \leftarrow \mathcal{K} \cup \{\top \sqsubseteq \downarrow z.S^z\}$
 4: $V_{LS}(z) \leftarrow S^z$
 5: for each $q \in Q$ do
 6:     if $q = C(x)$ or $q = r(x, y)$, $z \neq x$ then
 7:         choose $q_1, q_2, \ldots, q_n \in Q$ with
              $q_1 = r_1(z, y_1), q_2 = r_2(y_1, y_2),$
              $\ldots, q_n = r_n(y_{n-1}, x)$
 8:         for $1 \leq i \leq n$ do
 9:             absorbRT($q_i, V_{LS}, \mathcal{K}$)
10:         end for
11:     end if
12:     if $q = C(x)$ then
13:         absorbCT($C(x), V_{LS}, \mathcal{K}$)
14:         $z \leftarrow x$
15:     end if
16:     if $q = r(x, y)$ then
17:         absorbRT($r(x, y), V_{LS}, \mathcal{K}$)
18:         $z \leftarrow y$
19:     end if
20: end for
21: $S^{z_1 \ldots z_m z} \leftarrow V_{LS}(z)$
22: $\mathcal{K} \leftarrow \mathcal{K} \cup \{S^{z_1 \ldots z_m z} \sqsubseteq \bot\}$
```

Algorithm 2 absorbCT($C(x), V_{LS}, \mathcal{K}$)

```
1: $S^{x_1 \ldots x \ldots x_n} \leftarrow V_{LS}(x)$
2: $F^x_C \leftarrow$ fresh atomic concept
3: $S_C^{x_1 \ldots x \ldots x_n} \leftarrow$ fresh query state concept
4: $\mathcal{K} \leftarrow \mathcal{K} \cup \{S^{x_1 \ldots x \ldots x_n} \sqsubseteq \neg C \sqcup F^x_C\}$
5: $\mathcal{K} \leftarrow \mathcal{K} \cup \{S^{x_1 \ldots x \ldots x_n} \sqcap F^x_C \sqsubseteq S_C^{x_1 \ldots x \ldots x_n}\}$
6: $V_{LS}(x) \leftarrow S_C^{x_1 \ldots x \ldots x_n}$
```

Algorithm 3 absorbRT($r(x, y), V_{LS}, \mathcal{K}$)

```
 1: $S^{x_1 \ldots x \ldots x_n} \leftarrow V_{LS}(x)$
 2: $S_r^{x_1 \ldots x \ldots x_n} \leftarrow$ fresh query state concept
 3: $\mathcal{K} \leftarrow \mathcal{K} \cup \{S^{x_1 \ldots x \ldots x_n} \sqsubseteq \forall r.S_r^{x_1 \ldots x \ldots x_n}\}$
 4: if $V_{LS}(y)$ is undefined then
 5:     $S^y \leftarrow$ fresh query state concept
 6:     $\mathcal{K} \leftarrow \mathcal{K} \cup \{S_r^{x_1 \ldots x \ldots x_n} \sqsubseteq \downarrow y.S^y\}$
 7:     $V_{LS}(y) \leftarrow S^y$
 8: end if
 9: $S^{y_1 \ldots y \ldots y_m} \leftarrow V_{LS}(y)$
10: $S^{z_1 \ldots z_k} \leftarrow$ fresh query state concept with
        $z_1 \ldots z_k = x_1 \ldots x \ldots x_n y_1 \ldots y \ldots y_m$
11: $\mathcal{K} \leftarrow \mathcal{K} \cup$
        $\{S_r^{x_1 \ldots x \ldots x_n} \sqcap S^{y_1 \ldots y \ldots y_m} \sqsubseteq S^{z_1 \ldots z_k}\}$
12: $V_{LS}(y) \leftarrow S^{z_1 \ldots z_k}$
```

ensuring a correct blocking. First, we discuss the required rule extensions and define the notion of variable mappings:

Definition 1 (Variable Mapping). *A* variable mapping μ *is a (partial) function from variable names to nodes and we refer to the set of elements on which μ is defined as the* domain*, written* $\mathsf{dom}(\mu)$*, of μ. We say that two variable mappings μ_1 and μ_2 are* compatible *if $\mu_1(x) = \mu_2(x)$ for all $x \in \mathsf{dom}(\mu_1) \cap \mathsf{dom}(\mu_2)$.*

For an extended completion graph $G = (V, E, \mathcal{L}, \dot{\neq}, \mathcal{M})$ and $v \in V$, we denote with $\mathcal{M}(C, v)$ the sets of variable mappings that are associated with a concept C in $\mathcal{L}(v)$.

The $\downarrow$-rule creates and associates variable mappings with concept facts in the completion graph, which we then propagate to other concept facts w.r.t. the axioms from the query absorption by using the extensions of expansion rules depicted in Table 1. In particular, the application of the $\forall$-rule to a concept fact $\forall r.C(v)$ now also propagates mappings that are associated with $\forall r.C(v)$ to the concept C in the labels of the r-neighbours. If complex roles have to be handled, one can, for example, use an unfolding of the universal restriction according to the automata for role inclusion axioms [9].

The remaining rules of Table 1 handle the (lazy) unfolding of the new query state concepts in node labels. Please note that the standard unfolding rules for simple atomic

Table 1. Tableau rule extensions for creating and propagating variable mappings

$\downarrow$-rule:	if	$\downarrow x.C \in \mathcal{L}(v)$, v not indirectly blocked, and $C \notin \mathcal{L}(v)$ or $\{x \mapsto v\} \notin \mathcal{M}(C, v)$
	then	$\mathcal{L}(v) = \mathcal{L}(v) \cup \{C\}$ and $\mathcal{M}(C, v) = \mathcal{M}(C, v) \cup \{\{x \mapsto v\}\}$
$\forall$-rule:	if	$\forall r.C \in \mathcal{L}(v)$, v not indirectly blocked, there is an r-neighbour w of v with $C \notin \mathcal{L}(w)$ or $\mathcal{M}(\forall r.C, v) \nsubseteq \mathcal{M}(C, w)$
	then	$\mathcal{L}(w) = \mathcal{L}(w) \cup \{C\}$ and $\mathcal{M}(C, w) = \mathcal{M}(C, w) \cup \mathcal{M}(\forall r.C, v)$
$\sqsubseteq_1$-rule:	if	$S^{x_1 \ldots x_n} \sqsubseteq C \in \mathcal{K}$, $S^{x_1 \ldots x_n} \in \mathcal{L}(v)$, v not indirectly blocked, and $C \notin \mathcal{L}(v)$ or $\mathcal{M}(S^{x_1 \ldots x_n}, v) \nsubseteq \mathcal{M}(C, v)$
	then	$\mathcal{L}(v) = \mathcal{L}(v) \cup \{C\}$ and $\mathcal{M}(C, v) = \mathcal{M}(C, v) \cup \mathcal{M}(S^{x_1 \ldots x_n}, v)$
$\sqsubseteq_2$-rule:	if	$S^{x_1 \ldots x_n} \sqcap A \sqsubseteq C \in \mathcal{K}$, $\{S^{x_1 \ldots x_n}, A\} \subseteq \mathcal{L}(v)$, v not indirectly blocked, and $\mathcal{M}(S^{x_1 \ldots x_n}, v) \nsubseteq \mathcal{M}(C, v)$
	then	$\mathcal{L}(v) = \mathcal{L}(v) \cup \{C\}$ and $\mathcal{M}(C, v) = \mathcal{M}(C, v) \cup \mathcal{M}(S^{x_1 \ldots x_n}, v)$
$\sqsubseteq_3$-rule:	if	$S_1^{x_1 \ldots x_n} \sqcap S_2^{y_1 \ldots y_m} \sqsubseteq C \in \mathcal{K}$, $\{S_1^{x_1 \ldots x_n}, S_2^{y_1 \ldots y_m}\} \subseteq \mathcal{L}(v)$, v not indirectly blocked, and $(\mathcal{M}(S_1^{x_1 \ldots x_n}, v) \bowtie \mathcal{M}(S_2^{y_1 \ldots y_m}, v)) \nsubseteq \mathcal{M}(C, v)$
	then	$\mathcal{L}(v) = \mathcal{L}(v) \cup \{C\}$ and $\mathcal{M}(C, v) = \mathcal{M}(C, v) \cup (\mathcal{M}(S_1^{x_1 \ldots x_n}, v) \bowtie \mathcal{M}(S_2^{y_1 \ldots y_m}, v))$

concepts are still necessary, i.e., C has to be added to a node label for axioms of the form $A \sqsubseteq C$ and $A_1 \sqcap A_2 \sqsubseteq C$ if A or A_1 and A_2 are present. In contrast, the new unfolding rules are only applied if at least one concept on the left-hand side is a query state concept and they additionally also propagate associated variable mappings to C. More precisely, if the query state concept $S^{x_1 \ldots x_n}$ is in the label of a node v and we have the variable mappings M associated with this fact, then we add C for an axiom of the form $S^{x_1 \ldots x_n} \sqsubseteq C \in \mathcal{K}$ and we associate M also with $C(v)$ (cf. $\sqsubseteq_1$-rule). For an axiom of the form $S^{x_1 \ldots x_n} \sqcap A \sqsubseteq C$, we only add C and propagate the mappings to C if also the atomic concept A is in the label (cf. $\sqsubseteq_2$-rule). Finally, the $\sqsubseteq_3$-rule handles binary inclusion axioms, where both concepts on the left-hand side are query state concepts, by propagating the join of the associated variable mappings to the implied concept.

Definition 2 (Variable Mapping Join). *A variable mapping $\mu_1 \cup \mu_2$ is defined by setting $(\mu_1 \cup \mu_2)(x) = \mu_1(x)$ if $x \in \mathsf{dom}(\mu_1)$, and $(\mu_1 \cup \mu_2)(x) = \mu_2(x)$ otherwise. The* join *$\mathcal{M}_1 \bowtie \mathcal{M}_2$ between the sets of variable mappings $\mathcal{M}_1$ and $\mathcal{M}_2$ is defined as follows:*

$$\mathcal{M}_1 \bowtie \mathcal{M}_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \mathcal{M}_1, \mu_2 \in \mathcal{M}_2 \text{ and } \mu_1 \text{ is compatible with } \mu_2\}.$$

By applying the rules of Table 1 (in addition to the standard tableau rules) for a knowledge base that is extended by the axioms from the query absorption, we get associations of variable mappings with query state concepts such that they indicate which parts of a query (and how these parts) are satisfied in the completion graph.

Example 4 (Example 2 cont.). Assume we extend $\mathcal{K}_1 = \{A(a), A \sqsubseteq \exists t.B, B \sqsubseteq \exists r.A, t \sqsubseteq s^-, r \sqsubseteq s^-\}$ with the axioms from absorbing Q_1 in Figure 2 and test the consistency with a tableau algorithm extended by the rules of Table 1. We observe that the constructed completion graph contains a clash and, consequently, Q_1 is entailed (cf. Figure 3). More precisely, we create a node for the individual a and add A to its node label (due to $A(a)$). Now, we alternately create t- and r-successors (due to $A \sqsubseteq \exists t.B$ and $B \sqsubseteq \exists r.A$), where the t-successors are labelled with B and the r-successors with A. Due to $t \sqsubseteq s^-$ and $r \sqsubseteq s^-$, we add s^- to each edge label. It is obvious to see that the folding $\exists(t \sqcap s^-).\exists(r \sqcap s^-).\top$ of Q_1 (cf. Example 1 and Figure 1) is satisfied for each node that instantiates A.

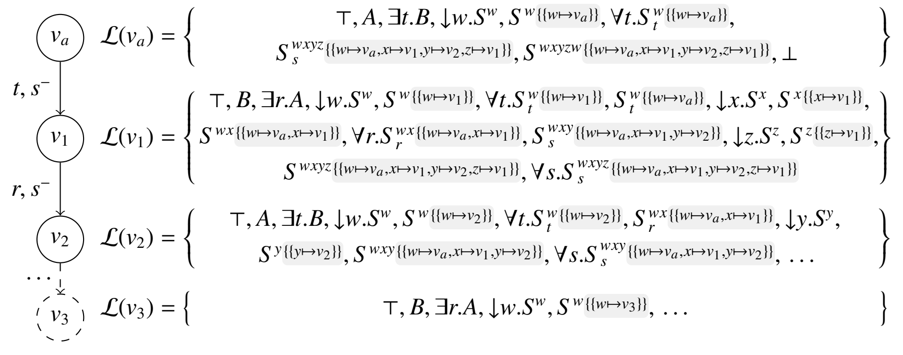

v_a
$\mathcal{L}(v_a) = \left\{ \top, A, \exists t.B, \downarrow w.S^w, S^{w\{\{w\mapsto v_a\}\}}, \forall t.S_t^{w\{\{w\mapsto v_a\}\}}, S_s^{wxyz\{\{w\mapsto v_a, x\mapsto v_1, y\mapsto v_2, z\mapsto v_1\}\}}, S^{wxyzw\{\{w\mapsto v_a, x\mapsto v_1, y\mapsto v_2, z\mapsto v_1\}\}}, \bot \right\}$
t, s^-
v_1
$\mathcal{L}(v_1) = \left\{ \top, B, \exists r.A, \downarrow w.S^w, S^{w\{\{w\mapsto v_1\}\}}, \forall t.S_t^{w\{\{w\mapsto v_1\}\}}, S_t^{w\{\{w\mapsto v_a\}\}}, \downarrow x.S^x, S^{x\{\{x\mapsto v_1\}\}}, S^{wx\{\{w\mapsto v_a, x\mapsto v_1\}\}}, \forall r.S_r^{wx\{\{w\mapsto v_a, x\mapsto v_1\}\}}, S_s^{wxy\{\{w\mapsto v_a, x\mapsto v_1, y\mapsto v_2\}\}}, \downarrow z.S^z, S^{z\{\{z\mapsto v_1\}\}}, S^{wxyz\{\{w\mapsto v_a, x\mapsto v_1, y\mapsto v_2, z\mapsto v_1\}\}}, \forall s.S_s^{wxyz\{\{w\mapsto v_a, x\mapsto v_1, y\mapsto v_2, z\mapsto v_1\}\}} \right\}$
r, s^-
v_2
$\mathcal{L}(v_2) = \left\{ \top, A, \exists t.B, \downarrow w.S^w, S^{w\{\{w\mapsto v_2\}\}}, \forall t.S_t^{w\{\{w\mapsto v_2\}\}}, S_r^{wx\{\{w\mapsto v_a, x\mapsto v_1\}\}}, \downarrow y.S^y, S^{y\{\{y\mapsto v_2\}\}}, S^{wxy\{\{w\mapsto v_a, x\mapsto v_1, y\mapsto v_2\}\}}, \forall s.S_s^{wxy\{\{w\mapsto v_a, x\mapsto v_1, y\mapsto v_2\}\}}, \ldots \right\}$
$\ldots$
v_3
$\mathcal{L}(v_3) = \left\{ \top, B, \exists r.A, \downarrow w.S^w, S^{w\{\{w\mapsto v_3\}\}}, \ldots \right\}$

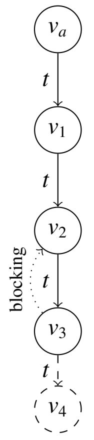
Fig. 3. Clashed completion graph for Example 4 with propagated variable mappings

Due to $\top \sqsubseteq \downarrow w.S^w$ from the absorption, we add S^w to each node label and associate S^w with a mapping from w to the node. In particular, for v_a representing the individual a, we associate $\{w \mapsto v_a\}$ with S^w. Note that $\{w \mapsto v_a\} \in \mathcal{M}(S^w, v_a)$ is shown as $S^{w\{\{w\mapsto v_a\}\}}$ in Figure 3, i.e., we list the set of associated mappings as a second super-script highlighted in grey. To satisfy the axiom $S^w \sqsubseteq \forall t.S_t^w$, we unfold S^w to $\forall t.S_t^w$ and we also keep the variable mappings, i.e., we have $\{w \mapsto v_a\} \in \mathcal{M}(\forall t.S_t^w, v_a)$. Now, the application of the $\forall$-rule propagates $\{w \mapsto v_a\}$ to $S_t^w \in \mathcal{L}(v_1)$. There, we unfold S_t^w to the binder concept for x, for which then the $\downarrow$-rule creates a new variable mapping $\{x \mapsto v_1\}$ that is joined by the $\sqsubseteq_3$-rule with $\{w \mapsto v_a\}$ such that we have $\{w \mapsto v_a, x \mapsto v_1\} \in \mathcal{M}(S^{wx}, v_1)$. These steps are repeated until we have $\{w \mapsto v_a, x \mapsto v_1, y \mapsto v_2, z \mapsto v_1\} \in \mathcal{M}(S_s^{wxyz}, v_a)$. Since $\{w \mapsto v_a\}$ is compatible with $\{w \mapsto v_a, x \mapsto v_1, y \mapsto v_2, z \mapsto v_1\}$, the $\sqsubseteq_3$-rule adds the latter variable mapping to $\mathcal{M}(S^{wxyzw}, v_a)$. Finally, the $\sqsubseteq_1$-rule adds $\bot$ to $\mathcal{L}(v_a)$. Since all facts and variable mappings are derived deterministically, no non-deterministic alternatives have to be evaluated and entailment of Q_1 is correctly determined.

As one can see from the example, the variable mappings associated with query state concepts directly correspond to foldings of the query. In particular, variables that are mapped to the same node correspond to the folding where the corresponding variables are identified. In addition, if a variable is mapped to a nominal node, then the mapping basically represents the "folding" that is obtained by replacing the variable with the associated nominal/individual (and folding up the remaining terms).

Without further optimisations, we create new bindings for every node and, due to complex roles and/or nominals, variable mappings might be propagated arbitrarily far through a completion graph. At first sight, this seems problematic for blocking. The correspondence with foldings, however, helps us to find a suitable extension of the typically used pairwise blocking technique [9] defined as follows:

Definition 3 (Pairwise Blocking). *Let $G = (V, E, \mathcal{L}, \dot{\neq}, \mathcal{M})$ be a completion graph. We say that a node v with predecessor v' is* directly blocked *if there exists an ancestor node w of v with predecessor w' such that (1) v, v', w, w' are all blockable, (2) w, w' are not blocked, (3) $\mathcal{L}(v) = \mathcal{L}(w)$ and $\mathcal{L}(v') = \mathcal{L}(w')$, and (4) $\mathcal{L}(\langle v', v\rangle) = \mathcal{L}(\langle w', w\rangle)$. A node*



$$\mathcal{L}(v_a) = \left\{ \top, A, \exists t.A, \downarrow w.S^w, S^{w\{\{w\mapsto v_a\}\}}, \forall t.S_t^{w\{\{w\mapsto v_a\}\}} \right\}$$

$$\mathcal{L}(v_1) = \left\{ \begin{array}{c} \top, A, \exists t.A, \downarrow w.S^w, S^{w\{\{w\mapsto v_1\}\}}, \forall t.S_t^{w\{\{w\mapsto v_1\},\{w\mapsto v_a\}\}}, \\ S_t^{w\{\{w\mapsto v_a\}\}}, \downarrow x.S^x, S^{x\{\{x\mapsto v_1\}\}}, \\ S^{wx\{\{w\mapsto v_a, x\mapsto v_1\}\}}, \forall r.S_r^{wx\{\{w\mapsto v_a, x\mapsto v_1\}\}} \end{array} \right\}$$

$$\mathcal{L}(v_2) = \left\{ \begin{array}{c} \top, A, \exists t.A, \downarrow w.S^w, S^{w\{\{w\mapsto v_2\}\}}, \forall t.S_t^{w\{\{w\mapsto v_2\},\{w\mapsto v_1\},\{w\mapsto v_a\}\}}, \\ S_t^{w\{\{w\mapsto v_1\},\{w\mapsto v_a\}\}}, \downarrow x.S^x, S^{x\{\{x\mapsto v_2\}\}}, \\ S^{wx\{\{w\mapsto v_a, x\mapsto v_2\},\{w\mapsto v_1, x\mapsto v_2\}\}}, \forall r.S_r^{wx\{\{w\mapsto v_a, x\mapsto v_2\},\{w\mapsto v_1, x\mapsto v_2\}\}} \end{array} \right\}$$

$$\mathcal{L}(v_3) = \left\{ \begin{array}{c} \top, A, \exists t.A, \downarrow w.S^w, S^{w\{\{w\mapsto v_3\}\}}, \forall t.S_t^{w\{\{w\mapsto v_3\},\{w\mapsto v_2\},\{w\mapsto v_1\},\{w\mapsto v_a\}\}}, \\ S_t^{w\{\{w\mapsto v_2\},\{w\mapsto v_1\},\{w\mapsto v_a\}\}}, \downarrow x.S^x, S^{x\{\{x\mapsto v_3\}\}}, \\ S^{wx\{\{w\mapsto v_a, x\mapsto v_3\},\{w\mapsto v_1, x\mapsto v_3\},\{w\mapsto v_2, x\mapsto v_3\}\}}, \forall r.S_r^{wx\{\{w\mapsto v_a, x\mapsto v_3\},\{w\mapsto v_1, x\mapsto v_3\},\{w\mapsto v_2, x\mapsto v_3\}\}} \end{array} \right\}$$

Fig. 4. Expansion blocked completion graph for Example 5 with variable mappings

is indirectly blocked *if it has an ancestor node that is directly blocked, and a node is blocked if it is directly or indirectly blocked.*

The query state concepts, which track how much of the query is satisfied, are already part of the concept labels. Hence, it remains to check whether the query is analogously satisfied (i.e., same foldings must exist) by, roughly speaking, checking whether the variable mappings have been propagated in the same way between the blocking node, its predecessor and (related) nominal nodes and between the blocked node, its predecessor and (related) nominal nodes. Note that a mapping μ and the query state concepts with which μ is associated capture which query parts are already satisfied. Query state concepts that are associated with mappings that are compatible with μ correspond to states where fewer or additional query parts are satisfied. The following notion captures such related query state concepts for a mapping μ and a node v of a completion graph:

Definition 4. *Let* $G = (V, E, \mathcal{L}, \dot{\neq}, \mathcal{M})$ *be a completion graph. For* $v \in V$ *and a mapping* μ*, we set* $\mathsf{states}(v, \mu) = \{C \in \mathcal{L}(v) \mid \mu_v \in \mathcal{M}(C, v)$ *is compatible with* $\mu\}$.

Note that we do not limit states to query state concepts only to enable more absorption optimisations (see [18] for details). We formally capture (query state) concepts associated with a mapping and their relation to blocking with the notion of *analogous propagation blocking* and *witness mappings*:

Definition 5 (Analogous Propagation Blocking). *Let* $G = (V, E, \mathcal{L}, \dot{\neq}, \mathcal{M})$ *be a completion graph and* $o_1, ..., o_n \in V$ *all the nominal nodes in G. We say that a node v with predecessor* v' *is* directly blocked *by w with predecessor* w' *if v is pairwise blocked by w and, for each mapping* $\mu \in \mathcal{M}(C, v) \cup \mathcal{M}(C, v') \cup \mathcal{M}(C, o_1) \cup ... \cup \mathcal{M}(C, o_n), C \in \mathcal{L}(v) \cup \mathcal{L}(v') \cup \mathcal{L}(o_1) \cup ... \cup \mathcal{L}(o_n)$*, there exists a* witness mapping $\mu' \in \mathcal{M}(D, w) \cup \mathcal{M}(D, w') \cup \mathcal{M}(D, o_1) \cup ... \cup \mathcal{M}(D, o_n), D \in \mathcal{L}(w) \cup \mathcal{L}(w') \cup \mathcal{L}(o_1) \cup ... \cup \mathcal{L}(o_n)$ *and vice versa such that* $\mathsf{states}(v, \mu) = \mathsf{states}(w, \mu')$, $\mathsf{states}(v', \mu) = \mathsf{states}(w', \mu')$*, and* $\mathsf{states}(o_i, \mu) = \mathsf{states}(o_i, \mu')$ *for* $1 \leq i \leq n$.

Table 2. Witness mappings for testing analogous propagation blocking for Example 5

μ	μ'	$\mathsf{states}(v_3,\mu) = \mathsf{states}(v_2,\mu')$	$\mathsf{states}(v_2,\mu) = \mathsf{states}(v_1,\mu')$
$\{w \mapsto v_3\}$	$\{w \mapsto v_2\}$	$\{S^w, \forall t.S^w_t, S^x\}$	$\{S^x\}$
$\{w \mapsto v_2\}$	$\{w \mapsto v_1\}$	$\{\forall t.S^w_t, S^w_t, S^x, S^{wx}, \forall r.S^{wx}_r\}$	$\{S^w, \forall t.S^w_t, S^x\}$
$\{w \mapsto v_1\}, \{w \mapsto v_a\}$	$\{w \mapsto v_a\}$	$\{\forall t.S^w_t, S^w_t, S^x, S^{wx}, \forall r.S^{wx}_r\}$	$\{\forall t.S^w_t, S^w_t, S^x, S^{wx}, \forall r.S^{wx}_r\}$
$\{x \mapsto v_3\}$	$\{x \mapsto v_2\}$	$\{S^w, \forall t.S^w_t, S^w_t, S^x, S^{wx}, \forall r.S^{wx}_r\}$	$\{S^w, \forall t.S^w_t, S^w_t\}$
$\{w \mapsto v_a, x \mapsto v_3\}$, $\{w \mapsto v_1, x \mapsto v_3\}$	$\{w \mapsto v_a, x \mapsto v_2\}$	$\{\forall t.S^w_t, S^w_t, S^x, S^{wx}, \forall r.S^{wx}_r\}$	$\{\forall t.S^w_t, S^w_t\}$
$\{w \mapsto v_2, x \mapsto v_3\}$	$\{w \mapsto v_1, x \mapsto v_2\}$	$\{\forall t.S^w_t, S^w_t, S^x, S^{wx}, \forall r.S^{wx}_r\}$	$\{S^w, \forall t.S^w_t\}$
$\{x \mapsto v_2\}$	$\{x \mapsto v_1\}$	$\{S^w, \forall t.S^w_t, S^w_t\}$	$\{S^w, \forall t.S^w_t, S^w_t, S^x, S^{wx}, \forall r.S^{wx}_r\}$
$\{w \mapsto v_a, x \mapsto v_2\}$, $\{w \mapsto v_1, x \mapsto v_2\}$	$\{w \mapsto v_a, x \mapsto v_1\}$	$\{\forall t.S^w_t, S^w_t\}$	$\{\forall t.S^w_t, S^w_t, S^x, S^{wx}, \forall r.S^{wx}_r\}$

Example 5 (Example 2 cont.). For testing entailment of Q_1 over $\mathcal{K}_2 = \{A(a),\ A \sqsubseteq \exists t.A,\ t \circ t \sqsubseteq t\}$, we can capture the transitivity of t by extending the axioms of Figure 2 with $S^w_t \sqsubseteq \forall t.S^w_t$ (cf. [9]). For the resulting axioms, the tableau algorithm creates a completion graph as depicted in Figure 4, where the query is not entailed. Due to the axiom $A \sqsubseteq \exists t.A$, the tableau algorithm successively builds t-successors until blocking is established. Note that new variable mappings are created for all nodes and all mappings are propagated to all descendants due to the transitive role t. Hence, we not only have mappings with new bindings for each new successor, but also an increasing number of mappings. Nevertheless, v_3 is already directly blocked by v_2 using analogous propagation blocking since all pairwise blocking conditions are satisfied (e.g., $\mathcal{L}(v_3) = \mathcal{L}(v_2)$, $\mathcal{L}(v_2) = \mathcal{L}(v_1)$) and we have for each variable mapping a witness mapping as shown in Table 2. For example, for the mapping $\{w \mapsto v_3\}$, we have $\mathsf{states}(v_3, \{w \mapsto v_3\}) = \{S^w, \forall t.S^w_t, S^x\}$ and $\mathsf{states}(v_2, \{w \mapsto v_3\}) = \{S^x\}$ due to the compatible mappings $\{x \mapsto v_3\}$ and $\{x \mapsto v_2\}$, respectively (cf. first row of Table 2). A witness for $\{w \mapsto v_3\}$ is $\{w \mapsto v_2\}$ since $\mathsf{states}(v_2, \{w \mapsto v_2\}) = \{S^w, \forall t.S^w_t, S^x\}$ and $\mathsf{states}(v_1, \{w \mapsto v_2\}) = \{S^x\}$.

To avoid considering all nominal nodes in blocking tests, one could obtain restricted sets of relevant nominal nodes by "remembering" nominal nodes over which variable mappings have been propagated, by tracking the usage of nominals for descendants or by indexing variable mappings propagated over nominal nodes.

4 Implementation and Experiments

The presented query entailment checking approach is implemented in the tableau-based reasoning system Konclude [20], which supports the DL $\mathcal{SROIQ}$ with nominal schemas, i.e., an extension of the nominal constructor by variables for natively representing rule-based knowledge in ontologies. Axioms with nominal schemas are also absorbed in Konclude such that variable bindings are appropriately propagated through the completion graph [19], which we reuse to some extent for the query entailment checking extension. The implementation and data for the experiments are available online [18].

At the moment, Konclude may not terminate for $\mathcal{SROIQ}$ ontologies if the absorption of the query leads to propagations over new nominal nodes. However, this does not

Table 3. Statistics for evaluated ontologies with query entailment checking (EC) times in seconds

Ontology	DL	#Axioms	#C	#P	#I	#Assertions	EC avg/max [s]
DMKB	$\mathcal{SROIQ}$	$4,945$	697	177	653	$1,500$	0.31 / 1.34
Family	$\mathcal{SROIQ}(\mathcal{D})$	317	61	87	405	$1,523$	180.33 / ≥300.00
Finance$_{\setminus\mathcal{D}}$	$\mathcal{ALCROIQ}$	$1,391$	323	247	$2,466$	$2,809$	0.09 / 0.27
FMA3.1$_{\setminus\mathcal{D}}$	$\mathcal{ALCOIN}$	$86,898$	$83,284$	122	$232,647$	$501,220$	0.11 / 0.78
GeoSkills$_{\setminus\mathcal{D}}$	$\mathcal{ALCHOIN}$	738	603	23	$2,592$	$5,985$	0.04 / 0.35
OBI	$\mathcal{SROIQ}(\mathcal{D})$	$6,216$	$2,826$	116	167	235	0.08 / 0.05
UOBM(1)	$\mathcal{SHOIN}(\mathcal{D})$	206	69	44	$24,858$	$257,658$	0.73 / 6.69
Wine	$\mathcal{SHOIN}(\mathcal{D})$	643	214	31	367	903	0.08 / 0.32

seem problematic in practice. For example, the ORE2015 dataset [15] contains 1920 ontologies (with trivial ontologies already filtered out), but only 399 use all problematic language features (36 are $\mathcal{OIQ}$, 281 are $\mathcal{OIN}$, and 82 are $\mathcal{OIF}$). Konclude never applied the new nominal rule in the consistency checks for these 399 ontologies, but we terminated the reasoner (and, hence, the analysis of the new nominal generation) for 4 ontologies after reaching the time limit of 5 minutes. Even if new nominals are generated, it would further be required that the query propagates differently over new nominal and blockable nodes such that blocking cannot be established.

For evaluating (the limits of) our query entailment checking approach, we identified several interesting ontologies (i.e., ontologies that use most features of $\mathcal{SROIQ}$ with at least 100 individuals and for which standard reasoning tasks are difficult but processable by Konclude), such as DMKB, FMA, OBI, UOBM (cf. Table 3), and generated 50 non-trivial queries with several cycles for each ontology. In summary, the entailment for most queries can be decided in under one second (90% require less than 1s, 49% less than 0.1s) by using one core of a Dell PowerEdge R420 server with two Intel Xeon E5-2440 CPUs at 2.4 GHz and 144 GB RAM under a 64bit Ubuntu 16.04.5 LTS. However, there are queries that lead to many propagations (e.g., UOBM) and/or require many blocking checks (e.g., Family) and, consequently, such queries require significantly more time (e.g., 30 queries for the Family ontology reached the time limit of 5 minutes since complex roles lead to many propagations with non-trivial blocking tests). To further improve the performance, one could also use a representative propagation of variable mappings [19] for entailment checks and/or index more precisely which nodes constitute blocker candidates. Nevertheless, the extension of the presented query entailment checking approach to query answering (with appropriate reduction optimisations) is already able to outperform PAGOdA [22] on some of the non-trivial real-world queries from the PAGOdA evaluation (see [18] for details).

5 Conclusions

We presented a novel query entailment checking approach based on the well-known absorption optimisation that improves the reasoning performance for several more expressive Description Logics. The approach can nicely be integrated into state-of-the-art tableau-based systems and our implementation in Konclude shows encouraging results.

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. Blackburn, P., Seligman, J.: Hybrid languages. Journal of Logic, Language and Information **4**(3) (1995)
3. Calvanese, D., Eiter, T., Ortiz, M.: Answering regular path queries in expressive description logics: An automata-theoretic approach. In: Proc. 22nd AAAI Conf. on Artificial Intelligence (AAAI'07). pp. 391–396. AAAI Press (2007)
4. Glimm, B., Horrocks, I., Sattler, U.: Unions of conjunctive queries in $\mathcal{SHOQ}$. In: Proc. 11th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'08). pp. 252–262. AAAI Press (2008)
5. Glimm, B., Kazakov, Y., Kollia, I., Stamou, G.: Lower and upper bounds for SPARQL queries over OWL ontologies. In: Proc. 29th Conf. on Artificial Intelligence (AAAI'15). AAAI Press (2015)
6. Glimm, B., Lutz, C., Horrocks, I., Sattler, U.: Conjunctive query answering for the description logic SHIQ. J. of Artificial Intelligence Research **31**, 157–204 (2008)
7. Grädel, E.: Why are modal logics so robustly decidable? In: Paun, G., Rozenberg, G., Salomaa, A. (eds.) Current Trends in Theoretical Computer Science, Entering the 21th Century, vol. 2, pp. 393–408. World Scientific (2001)
8. Haarslev, V., Möller, R., Wessel, M.: Querying the semantic web with Racer + nRQL. In: Proc. KI-2004 Int. Workshop on Applications of Description Logics (2004)
9. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: Proc. 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'06). pp. 57–67. AAAI Press (2006)
10. Horrocks, I., Tessaris, S.: Querying the semantic web: a formal approach. In: Proc. 1st Int. Semantic Web Conf. (ISWC'02). pp. 177–191. Springer (2002)
11. 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)
12. Kollia, I., Glimm, B.: Optimizing SPARQL query answering over OWL ontologies. J. of Artificial Intelligence Research **48**, 253–303 (2013)
13. Ortiz, M., Calvanese, D., Eiter, T.: Data complexity of query answering in expressive description logics via tableaux. J. of Automated Reasoning **41**(1), 61–98 (2008)
14. Pan, J.Z., Thomas, E., Zhao, Y.: Completeness guaranteed approximation for OWL-DL query answering. In: Proceedings of the 22nd International Workshop on Description Logics (DL'09). vol. 477. CEUR (2009)
15. Parsia, B., Matentzoglu, N., Gonçalves, R.S., Glimm, B., Steigmiller, A.: The OWL reasoner evaluation (ORE) 2015 competition report. J. of Automated Reasoning **59**(4), 455–482 (2017)
16. Rudolph, S., Glimm, B.: Nominals, inverses, counting, and conjunctive queries or: Why infinity is your friend! J. of Artificial Intelligence Research **39**, 429–481 (2010)
17. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. J. of Web Semantics **5**(2), 51–53 (2007)
18. Steigmiller, A., Glimm, B.: Absorption-based query answering for expressive description logics – technical report. Tech. rep., Ulm University, Ulm, Germany (2019), available online at https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.090/Publikationen/2019/StGl2019-ABQA-TR-DL.pdf
19. Steigmiller, A., Glimm, B., Liebig, T.: Reasoning with nominal schemas through absorption. J. of Automated Reasoning **53**(4), 351–405 (2014)

20. Steigmiller, A., Liebig, T., Glimm, B.: Konclude: system description. J. of Web Semantics **27**(1) (2014)
21. Vardi, M.Y.: Why is modal logic so robustly decidable? In: Descriptive Complexity and Finite Models: Proceedings of a DIMACS Workshop. DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, vol. 31, pp. 149–184. Memoirs of the American Mathematical Society (1997)
22. Zhou, Y., Cuenca Grau, B., Nenov, Y., Kaminski, M., Horrocks, I.: PAGOdA: Pay-as-you-go ontology query answering using a datalog reasoner. J. of Artificial Intelligence Research **54**, 309–367 (2015)