Absorption-Based Query Answering 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 many knowledge bases 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 answering conjunctive queries that can directly be integrated into existing reasoning systems for expressive Description Logics. 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 outperforms existing systems even for queries that are restricted to the capabilities of these other systems.

1 Introduction

A distinguished feature of logic-based knowledge representation formalisms, such as Description Logics (DLs), is the ability to use automated reasoning techniques to access implicit knowledge of explicitly stated information. In particular, a DL knowledge base can be seen as a collection of explicitly stated information that describes a domain of interest, i.e., individuals/entities and their features. Roles are used to state the relationship between individuals, concepts represent sets of individuals with common characteristics, and axioms relate concepts or roles to each other, e.g., by specifying sub-concept relationships, or state facts about an individual/a pair of individuals. Since the DL $\mathcal{SROIQ}$ [10] is the logical underpinning of the second and current iteration of the well-known Web Ontology Language (OWL), its language features are often used in practice for modelling ontologies. Consequently, reasoning systems that support $\mathcal{SROIQ}$ are required to work with these ontologies. So far, most reasoners for expressive DLs, such as $\mathcal{SROIQ}$, are based on variants of tableau algorithms since they are easily extensible and adaptable to the expressive language features. Moreover, many developed optimisation techniques allow these systems to efficiently handle standard reasoning tasks (e.g., consistency checking, classification, instance retrieval, etc.) for

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

many real-world ontologies. To satisfy all user demands, more sophisticated reasoning tasks such as conjunctive query answering are also often required. Such queries consist of a conjunction of concept and role facts, where variables may be used in place of individuals. Such variables may be existentially quantified (aka non-distinguished variables) or answer variables (aka distinguished variables). For the answer variables, the reasoner has to deliver bindings to named individuals of the knowledge base such that the query, instantiated with the bindings, is entailed by the knowledge base. For existential variables, it is only required that there exists a binding to any, possibly anonymous individual in each model.

To the best of our knowledge, current reasoning systems support conjunctive queries for expressive DLs only with limitations. This is due to several reasons. First, decidability of conjunctive query entailment, to which query answering is typically reduced, is still open in $\mathcal{SROIQ}$. Second, while the decidability and the worst-case complexity has been shown for many sub-languages (e.g., [3,15,18]), the used techniques 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, [5,7,11]), which is not goal-directed and often requires many unnecessary entailment checks. Moreover, some of these reduction techniques 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 (conjunctive instance queries), existing approaches (e.g., [9,13,19]) are often impractical since they are based on the above described reduction to instance checking. Moreover, by only using existing reasoning systems as black-boxes, the possibility to optimise conjunctive query answering is limited. Recently, query answering has been improved by lower and upper bound optimisations that utilise model abstractions built by a reasoner [6] or delegate work to specialised procedures [16,25]. Furthermore, it is possible to determine for which queries the answers from specialised systems can be complete although not all used language features are completely handled [23]. However, the specialised procedures are still used as a black-box and delegating all work to them is not possible in general. Hence, practical conjunctive query answering techniques for expressive DLs are still needed.

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 tableau calculus. 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 answering* since it reminds of the absorption technique for nominal schemas [21]. The approach is correct and terminates for DLs for which decidability of conjunctive query answering is known (e.g., $\mathcal{SHIQ}$, $\mathcal{SHOQ}$). For the challenging combination of nominals, inverse roles, and number restrictions, termination is only guaranteed if a limited 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. In fact, we implemented the proposed technique in the reasoning system Konclude [22] with encouraging results.

Table 1. Core features of $\mathcal{SROIQ}$ ($\#M$ denotes the cardinality of the set M)

		Syntax	Semantics
Individuals:	individual	a	$a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
Roles:	atomic role	r	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
	inverse role	r^-	$\{\langle \gamma, \delta \rangle \mid \langle \delta, \gamma \rangle \in r^{\mathcal{I}}\}$
Concepts:	atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
	nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$
	top	$\top$	$\Delta^{\mathcal{I}}$
	bottom	$\bot$	$\emptyset$
	negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
	conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
	disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
	existential restriction	$\exists R.C$	$\{\delta \mid \exists \gamma \in C^{\mathcal{I}} : \langle \delta, \gamma \rangle \in R^{\mathcal{I}}\}$
	universal restriction	$\forall R.C$	$\{\delta \mid \langle \delta, \gamma \rangle \in R^{\mathcal{I}} \rightarrow \gamma \in C^{\mathcal{I}}\}$
	number restriction, $\bowtie \in \{\leqslant, \geqslant\}$	$\bowtie n\, R.C$	$\{\delta \mid \#\{\langle \delta, \gamma \rangle \in R^{\mathcal{I}} \text{ and } \gamma \in C^{\mathcal{I}}\} \bowtie n\}$
Axioms:	general concept inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
	role inclusion	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
	role chains	$R_1 \circ \ldots \circ R_n \sqsubseteq S$	$R_1^{\mathcal{I}} \circ \ldots \circ R_n^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
	concept assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
	role assertion	$R(a, b)$	$\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$
	equality assertion	$a \approx b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$

The paper is organised as follows: Section 2 gives a brief introduction into DLs and reasoning. Section 3 describes the absorption-based query entailment checking technique, for which reductions from query answering are sketched in Section 4. Section 5 discusses the implementation and evaluation results. Additional explanations, examples, and evaluation results can be found in an accompanying technical report [20].

2 Preliminaries

Due to space restrictions, we only give a brief introduction into DLs and reasoning techniques (see, e.g., [1], for more details).

2.1 Description Logics and Conjunctive Queries

The syntax of DLs is defined using a vocabulary consisting of countably infinite pairwise disjointed sets N_C of *atomic concepts*, N_R of *atomic roles*, and N_I of *individuals*. A role is either atomic or an *inverse role* r^-, $r \in N_R$. The syntax and semantics of complex *concepts* and *axioms* are defined in Table 1. Note that we omit the presentation of some features (e.g., datatypes) and restrictions (e.g., number restrictions may not use "complex roles", i.e., roles that occur on the right-hand side of role chains) for brevity. A knowledge base/ontology $\mathcal{K}$ is a finite set of axioms. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty *domain* $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$. We say that $\mathcal{I}$ *satisfies* a general concept inclusion (GCI) $C \sqsubseteq D$, written $\mathcal{I} \models C \sqsubseteq D$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ (analogously for other axioms as shown in Table 1). If $\mathcal{I}$ satisfies all axioms of a knowledge base $\mathcal{K}$, $\mathcal{I}$ is a *model* of $\mathcal{K}$ and $\mathcal{K}$ is *consistent/satisfiable* if it has a model.

A *conjunctive query* $Q(X, Y)$ consists of a set of *query terms* $q_1, \ldots, q_k$, where X denotes the tuple of answer variables, Y the tuple of existential variables (disjoint to X), and each q_i is either a concept term $C(z)$ or a role term $r(z_1, z_2)$ with $z, z_1, z_2 \in \mathsf{vars}(Q)$, where $\mathsf{vars}(Q)$ is the set of variable names occurring in $Q(X, Y)$. A *Boolean query* $Q(\langle\rangle, Y)$, short Q, is a query without answer variables. To simplify the handling of inverse roles, we consider $r(x, y)$ as equivalent to $r^-(y, x)$. For an interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ and a total function $\pi : \mathsf{vars}(Q) \mapsto \Delta^\mathcal{I}$, we say that π is a *match* for $\mathcal{I}$ and Q if, for every $C(z) \in Q$, $\pi(z) \in C^\mathcal{I}$ and, for every $r(z_1, z_2) \in Q$, $\langle \pi(z_1), \pi(z_2) \rangle \in r^\mathcal{I}$. We say that an n-ary tuple of the form $\langle a_1, \ldots, a_n \rangle$ with $a_1, \ldots, a_n$ individuals of $\mathcal{K}$ is an *answer* for $Q(\langle x_1, \ldots, x_n \rangle, Y)$ w.r.t. $\mathcal{K}$ if, for every model $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ of $\mathcal{K}$, there exists a match π for $\mathcal{I}$ and Q with $\pi(x_i) = a_i^\mathcal{I}$ for $1 \leq i \leq n$. If a query $Q(X, Y)$ ($Q(\langle\rangle, Y)$) has an answer (the empty answer $\langle\rangle$) w.r.t. $\mathcal{K}$, then we say that $\mathcal{K}$ *entails* Q and with *query answering* (*query entailment checking*) we refer to the reasoning task that computes all answers (the entailment of the empty answer). W.l.o.g. we use individual names only in nominal concepts and we assume that all variables are connected via role terms.

2.2 Tableau Algorithm

A tableau algorithm decides 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 expanded, clash-free completion graph. Cycle detection techniques such as *pairwise blocking* [10] 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

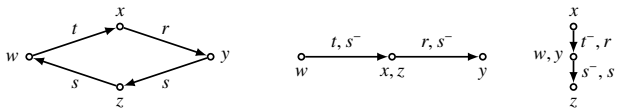

Fig. 1. Visualisation of the query of Example 1 and two possible foldings

[12] 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

Since query answering is typically reduced to query entailment checking, we first focus on a decision procedure for the latter. With the exception of role relationships between nominals/individuals, DLs allow only for expressing tree-shaped structures [8,24]. Even with nominals/individuals, forest-shaped models exists [18]. 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 Boolean 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 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 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.

$$\top \sqsubseteq \downarrow w.S^w \qquad S^w \sqsubseteq \forall t.S^w_t \qquad S^w_t \sqsubseteq \downarrow x.S^x$$
$$S^w_t \sqcap S^x \sqsubseteq S^{wx} \qquad S^{wx} \sqsubseteq \forall r.S^{wx}_r \qquad S^{wx}_r \sqsubseteq \downarrow y.S^y$$
$$S^{wx}_r \sqcap S^y \sqsubseteq S^{wxy} \qquad S^{wxy} \sqsubseteq \forall s.S^{wxy}_s \qquad S^{wxy}_s \sqsubseteq \downarrow z.S^z$$
$$S^{wxy}_s \sqcap S^z \sqsubseteq S^{wxyz} \qquad S^{wxyz} \sqsubseteq \forall s.S^{wxyz}_s \qquad S^{wxyz}_s \sqcap S^w \sqsubseteq S^{wxyzw} \qquad S^{wxyzw} \sqsubseteq \bot$$

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

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 it with $\{x \mapsto v\}$. Finally, for joining bindings, we extend the binary unfolding rule. For example, for an axiom $S_1 \sqcap S_2 \sqsubseteq C$ and $S_1, S_2 \in \mathcal{L}(v)$ associated with $\{x \mapsto v, y \mapsto w\}$ and $\{x \mapsto v, z \mapsto w\}$, respectively, we add C associated with the joined bindings $\{x \mapsto v, y \mapsto w, z \mapsto w\}$ to $\mathcal{L}(v)$. With these basic adaptations, we can capture the query in several simple types of axioms: $S_1 \sqsubseteq \downarrow x.S_2$ for creating bindings, $S_1 \sqsubseteq S_2$ and $S_1 \sqsubseteq \forall r.S_2$ for propagating bindings, and $S_1 \sqcap S_2 \sqsubseteq S_3$ for joining bindings, where $S_{(i)}$ are query state concepts and r is a role. 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

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 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: $S^{z_1 \cdots z_m z} \leftarrow V_{LS}(z)$
22: $\mathcal{K} \leftarrow \mathcal{K} \cup \{S^{z_1 \cdots z_m z} \sqsubseteq \bot\}$

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

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

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

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

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

Table 2. 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	$S^{x_1 \dots x_n} \sqsubseteq C \in \mathcal{K}$, $S^{x_1 \dots x_n} \in \mathcal{L}(v)$, v not indirectly blocked, and $C \notin \mathcal{L}(v)$ or $\mathcal{M}(S^{x_1 \dots x_n}, v) \nsubseteq \mathcal{M}(C, v)$
	then	$\mathcal{L}(v) = \mathcal{L}(v) \cup \{C\}$ and $\mathcal{M}(C, v) = \mathcal{M}(C, v) \cup \mathcal{M}(S^{x_1 \dots x_n}, v)$
$\sqsubseteq_2$-rule:	if	$S^{x_1 \dots x_n} \sqcap A \sqsubseteq C \in \mathcal{K}$, $\{S^{x_1 \dots x_n}, A\} \subseteq \mathcal{L}(v)$, v not indirectly blocked, and $\mathcal{M}(S^{x_1 \dots x_n}, v) \nsubseteq \mathcal{M}(C, v)$
	then	$\mathcal{L}(v) = \mathcal{L}(v) \cup \{C\}$ and $\mathcal{M}(C, v) = \mathcal{M}(C, v) \cup \mathcal{M}(S^{x_1 \dots x_n}, v)$
$\sqsubseteq_3$-rule:	if	$S_1^{x_1 \dots x_n} \sqcap S_2^{y_1 \dots y_m} \sqsubseteq C \in \mathcal{K}$, $\{S_1^{x_1 \dots x_n}, S_2^{y_1 \dots y_m}\} \subseteq \mathcal{L}(v)$, v not indirectly blocked, and $(\mathcal{M}(S_1^{x_1 \dots x_n}, v) \bowtie \mathcal{M}(S_2^{y_1 \dots y_m}, v)) \nsubseteq \mathcal{M}(C, v)$
	then	$\mathcal{L}(v) = \mathcal{L}(v) \cup \{C\}$ and $\mathcal{M}(C, v) = \mathcal{M}(C, v) \cup (\mathcal{M}(S_1^{x_1 \dots x_n}, v) \bowtie \mathcal{M}(S_2^{y_1 \dots y_m}, v))$

(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 examples, we always chose an adjacent next query term that contained 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 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 it 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 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 2.

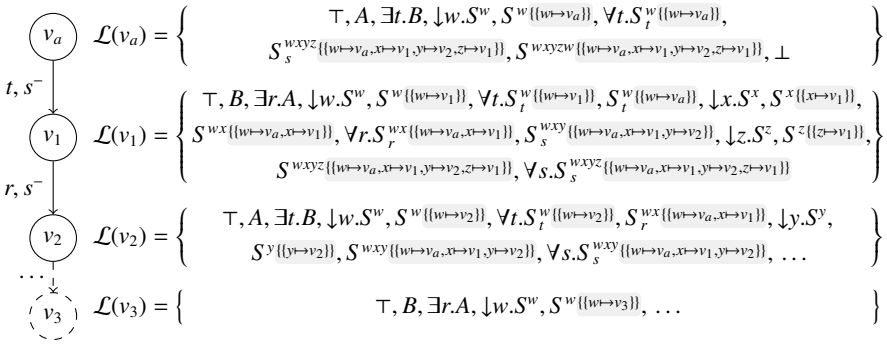

Fig. 3. Clashed completion graph for Example 4 with propagated variable mappings

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 [10].

The remaining rules of Table 2 handle the (lazy) unfolding of the new query state concepts in node labels. Please note that the standard unfolding rules for simple atomic 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, for a query state concept $S^{x_1 \ldots x_n} \in \mathcal{L}(v)$ with $\mathcal{M}(S^{x_1 \ldots x_n}, v) = M$ and an axiom $S^{x_1 \ldots x_n} \sqsubseteq C \in \mathcal{K}$, the $\sqsubseteq_1$-rule adds C to $\mathcal{L}(v)$ and associates it with M. 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 2 (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 2. 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.

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 $S^w \sqsubseteq \forall t.S^w_t$, we unfold S^w to $\forall t.S^w_t$ and we also keep the variable mappings, i.e., we have $\{w \mapsto v_a\} \in \mathcal{M}(\forall t.S^w_t, v_a)$. Now, the application of the $\forall$-rule propagates $\{w \mapsto v_a\}$ to $S^w_t \in \mathcal{L}(v_1)$. There, we unfold S^w_t to the binder concept for x, i.e., $\downarrow x.S^x$. Note that the unfolding would propagate the associated variable mapping(s) to the binder concept, but they are not depicted in the figures since they are not further used by the $\downarrow$-rule. In fact, the $\downarrow$-rule just creates a new variable mapping $\{x \mapsto v_1\}$ that is then 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 $\{w \mapsto v_a, x \mapsto v_1, y \mapsto v_2, z \mapsto v_1\} \in \mathcal{M}(S^{wxyz}_s, v_a)$. 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 $\mathcal{L}(v_a)$. (In the figures, we again omit the variable mappings that would be associated with $\bot$ due to the unfolding since they are not relevant.) 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 [10] 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 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 node 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

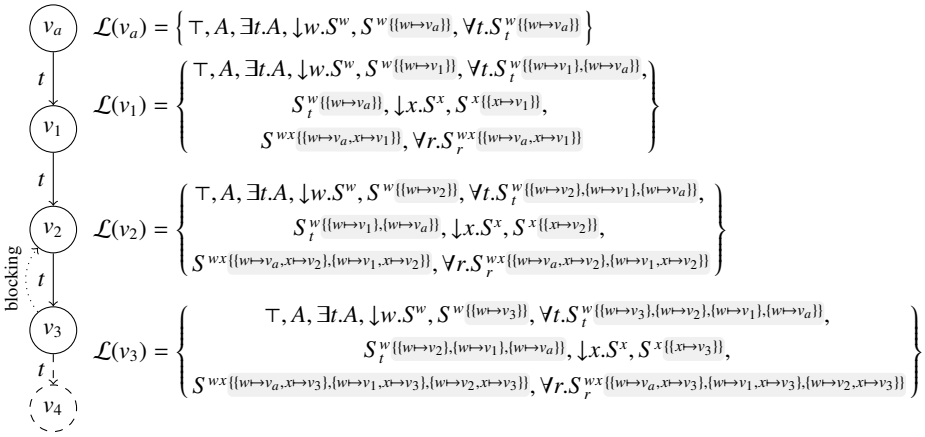

Fig. 4. Expansion blocked completion graph for Example 5 with variable mappings

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 [20] 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* $1 \leq i \leq n$.

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 $S_t^w \sqsubseteq \forall t.S_t^w$ (cf. [10]). 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 cyclic 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,

Table 3. Witness mappings for testing analogous propagation blocking for Example 5

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

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 3. For example, for the mapping $\{w \mapsto v_3\}$, we have $\mathsf{states}(v_3, \{w \mapsto v_3\}) = \{S^w, \forall t.S_t^w, S^x\}$ 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 3). A witness for $\{w \mapsto v_3\}$ is $\{w \mapsto v_2\}$ since $\mathsf{states}(v_2, \{w \mapsto v_2\}) = \{S^w, \forall t.S_t^w, S^x\}$ 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.

3.3 Correctness and Termination Sketches

As long as no new nominals are generated, a tableau algorithm with the presented extensions terminates since the sets of (query state) concepts that are used by analogous propagation blocking are bounded by the number of concepts occurring in the knowledge base and in the query absorption. Hence, we eventually have variable mappings that are associated with the same set of concepts for the nodes relevant for determining blocking. At the same time, analogous propagation blocking delays blocking sufficiently to guarantee that the completion graph is expanded enough to show (non-)entailment of the query. In addition, we observe that the query state concepts with their associated variable mappings correspond to concepts that represent satisfied (parts of) foldings of the query. Hence, correctness of the algorithm can be shown by transforming a fully expanded and clash-free completion graph with propagated variable mappings to a fully expanded and clash-free completion graph, where instead the correspondingly satisfied (sub-)concepts of folding are in the node labels and vice versa. Further details are provided in the accompanying technical report [20].

4 Optimized Query Answering Reduction

Instead of naively grounding conjunctive queries with answer variables (e.g., by adding nominal concept terms), leading to (exponentially) many query entailment checks, we can modify the presented absorption algorithm such that it delivers (candidates for) answers. For this, we extend the knowledge base by assertions of the form $O(a)$ for each individual a, where O is a fresh atomic concept that allows for distinguishing known and anonymous individuals. We then extend the query with concept terms of the form $O(x)$ for each answer variable x and absorb the query as presented in Section 3, but omit the implication of $\bot$ from the last query state concept. If the tableau algorithm succeeds to construct a fully expanded and clash-free completion graph, then the variable mappings that are propagated to and associated with the last query state concept in node labels encode the answer candidates. Moreover, by analysing whether variable mappings have been propagated deterministically, i.e., mappings that do not depend on non-deterministic decisions, we can already identify which of the candidates are certain answers. For the non-deterministically derived/propagated variable mappings, we have to verify the obtained answer candidates with corresponding query entailment checks. If most consequences of the knowledge base can be derived deterministically, then we often also get the answers by only extracting them from the propagated variable mappings, i.e., the approach mimics a one-pass behaviour for "relatively simple and mostly deterministic" ontologies.

The presented query answering approach can further be extended to exploit realisation results. In fact, query terms with only answer variables (and only atomic concepts) correspond to (atomic) concept and role instance retrieval queries, which are typically answered by the realisation service. Hence, we resolve these parts of a query via realisation and interpret the answers as an upper bound for the entire query. Then, we absorb the remaining part of the query by using restricted binder concepts for answer variables such that the $\downarrow$-rule only creates a binding if the binder concept is in the label of a node that represents an individual from the determined upper bound of the corresponding variable. This reduces the propagation work in completion graphs since only the remaining query terms have to be considered and only certain bindings are created.

5 Implementation and Experiments

We implemented the presented query answering approach into the tableau-based reasoning system Konclude [22], 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 [21], which we reuse to some extent for the query answering extension. A major difference is, however, that bindings for variables are now also created/allowed for anonymous individuals, i.e., blockable nodes in completion graphs, which requires the more sophisticated blocking technique. In addition, specialised binder concepts are used to be able to restrict the creation of bindings to determined candidates as described in Section 4. Furthermore, Konclude uses variants of completion graph caching techniques such that only those parts of completion graphs have to be constructed that are

Table 4. Statistics for evaluated ontologies with query entailment checking (EC) times in seconds

Ontology	DL	#Axioms	#C	#P	#I	#Assertions	#Q	EC avg/max [s]
DMKB	$\mathcal{SROIQ}$	4,945	697	177	653	1,500	50	0.30 / 1.08
Family	$\mathcal{SROIQ}(\mathcal{D})$	317	61	87	405	1,523	50	180.32 / ≥300.00
Finance$_{\setminus\mathcal{D}}$	$\mathcal{ALCROIQ}$	1,391	323	247	2,466	2,809	50	0.15 / 0.33
FMA3.1$_{\setminus\mathcal{D}}$	$\mathcal{ALCOIN}$	86,898	83,284	122	232,647	501,220	50	0.10 / 0.83
GeoSkills$_{\setminus\mathcal{D}}$	$\mathcal{ALCHOIN}$	738	603	23	2,592	5,985	50	0.13 / 0.25
OBI	$\mathcal{SROIQ}(\mathcal{D})$	6,216	2,826	116	167	235	50	0.06 / 0.34
UOBM(1)	$\mathcal{SHOIN}(\mathcal{D})$	206	69	44	24,858	257,658	50	0.77 / 7.12
Wine	$\mathcal{SHOIN}(\mathcal{D})$	643	214	31	367	903	50	0.08 / 0.29

(potentially) relevant for satisfiability tests, which has been adapted for query answering. The cached graphs are also indexed to quickly resolve candidates for answer variables. We further collect approximative statistics of how (often) concept and role facts are derived in the consistency test (e.g., whether they are derived (non-)deterministically and/or only for nominal nodes) in order to absorb the query terms in an order that ideally leads to few and cheap propagations (e.g., by preferring propagations over role terms with few and mostly deterministically derived instances).

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 seem problematic in practice. For example, the ORE2015 dataset [17] contains 1920 ontologies (with trivial ontologies already filtered out), but only 399 use all problematic language features (36 with unqualified, 281 with qualified, and 82 with functional number restrictions). 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 have to be generated repeatedly by the tableau algorithm, 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) the presented query entailment checking approach,[1] we generated and/or hand-crafted 400 cyclic and non-trivial queries for interesting ontologies, such as DMKB, FMA, OBI, UOBM [14], i.e., ontologies from well-known repositories that use many features of $\mathcal{SROIQ}$ and have at least 100 individuals. Table 4 shows some metrics for these ontologies as well as the average and maximum query entailment checking times (last column). Note that the columns #C, #P, #I, and #Q denote the number of classes, properties, individuals, and queries and that the assertions are not counted to the number of axioms. In summary (see [20] for details), the entailment for most queries (90%) can be decided in under one second 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, Konclude reached the time limit of 5 minutes for several queries of the Family ontology since complex roles caused propagations to many nodes such that blocking tests became quite involved.

[1] Source code, evaluation data, all results, and a Docker image (koncludeeval/abqa) for easily reproducing the evaluations are available online, e.g., at https://zenodo.org/record/3266160.

Table 5. Ontologies with evaluated query answering times in seconds

Ontology	DL	#Axioms	#Assertions	#Q	Konclude	OWL BGP	PAGOdA	Pellet
$\text{ChEMBL}_{1\%}$	$\mathcal{SRIQ}(\mathcal{D})$	3,171	2,884,896	6	0.3	1800.0	0.7	17.7
FLY	$\mathcal{SRI}$	20,715	1,606	11	4.7	3300.0	20.3	3300.0
LUBM(1)	$\mathcal{ALEHI}^{+}(\mathcal{D})$	93	100,543	35	1.2	16.8	5.4	11.2
$\text{Reactome}_{10\%}$	$\mathcal{SHIN}(\mathcal{D})$	600	1,314,640	7	0.4	2100.0	14.4	33.8
$\text{Uniprot}_{1\%}$	$\mathcal{ALCHOIQ}(\mathcal{D})$	608	1,112,441	13	0.3	3900.0	2.7	307.8
UOBM(1)	$\mathcal{SHIN}(\mathcal{D})$	246	257,658	20	6.6	6000.0	22.7	972.7
ALL				92	13.6	17116.8	66.1	4643.2

Due to the fact that many state-of-the-art reasoners such as HermiT [4] and Pellet [19] cannot even classify the Family ontology within 5 minutes, it can be seen as particularly difficult. It also has to be considered that typical real-world queries are often not Boolean and it is usually possible to identify individuals that could be affected by the query (e.g., from answer variables) such that only certain parts of completion graphs have to be considered for the remaining (entailment) computations (with appropriate completion graph caching techniques). To further improve the performance, one could, however, also use a representative propagation of variable mappings [21] for entailment checks and/or index more precisely which nodes constitute blocker candidates.

To further compare our query answering approach with existing systems, we used the ontologies and (non-trivial) queries from the PAGOdA evaluation [25] (cf. Table 5).[1] Note that we excluded trivial concept and role instance retrieval queries since they can be handled by (concept and role) realisation, for which already corresponding evaluations exist (see, e.g., [17]). Since these expressive ontologies easily become problematic for several reasoning systems, we used the smallest versions of the available datasets.

We compared the absorption-based query answering approach with OWL BGP [6], PAGOdA, and Pellet, which are systems that are based on fully-fledged $\mathcal{SROIQ}$ reasoners and, hence, principally capable of answering (restricted) conjunctive queries w.r.t. knowledge bases that are formulated with more expressive DLs. Note that OWL BGP as well as PAGOdA use a fully-fledged reasoner (usually HermiT) in form of a black-box, but try to reduce the calls to the reasoner with different (lower and upper bound) optimisations. OWL BGP can mostly be considered as an adapter that enables to answer conjunctive instance queries, whereas PAGOdA tries to delegate most of the workload to a more efficient datalog engine. We tried to separate the preprocessing (i.e., loading, consistency checking, classification, etc.) from the actual query answering time. Since we could not find a simple way to realise this for Pellet, we interpreted the fastest query answering response as preprocessing/preparation time. As far it has been configurable, we used only one core of the Dell PowerEdge R420 server for each reasoner to facilitate the comparison. Moreover, we used for each query a new instance of the reasoner with a time limit of 5 minutes. If a reasoner did not finish the preprocessing within the time limit, then we also interpreted the response time for the query answering task as a timeout, i.e., as 5 minutes.

The query answering times accumulated per ontology are depicted in Table 5. By considering the accumulated times over all ontologies (last row), one can say that Kon-

clude outperforms the other systems for the evaluated ontologies and queries. In fact, Konclude is faster than the other systems for all ontologies. Although PAGOdA can answer all queries for all evaluated ontologies, it is clearly slower than Konclude for $\text{Reactome}_{10\%}$ and the FLY ontology. For FLY, PAGOdA had to fall back to the fully-fledged reasoner, i.e., HermiT, for one query and these calls consumed most of the time. PAGOdA and Konclude returned the same answers for all queries,[2] whereas some answers are incomplete for OWL BGP and Pellet due to their restrictions w.r.t. existential variables. Pellet and especially OWL BGP are significantly slower than Konclude and PAGOdA. On the one hand, this is due to the fact that Pellet and HermiT cannot handle these ontologies as easily as Konclude and they do not have optimisations to delegate parts of the reasoning to more specialised systems (such as PAGOdA). On the other hand, our query answering is much "deeper integrated" into the reasoning system and, hence, it can better utilise the internal data structures and can profit more from corresponding (reduction) optimisations. Also preprocessing (i.e., loading, consistency checking, etc.) is significantly faster for Konclude than for the other systems (it required at most half of their time).

6 Conclusions

We presented a new query answering approach based on the well-known absorption optimisation that works well for several more expressive Description Logics and can nicely be integrated into state-of-the-art tableau-based reasoning systems. More precisely, the approach rewrites/absorbs a conjunctive query into several simple axioms such that minor extensions of the tableau algorithm appropriately create and propagate bindings for variables through completion graphs, which then basically encode the satisfied foldings of a query. Soundness, completeness, and termination is guaranteed as long as only a limited number of new nominals has to be introduced in the reasoning process, which seems always the case in practice.

The deep integration facilitates special optimisations that closely interact with other reasoning services and utilise the internal data structures of the reasoner, which results in a good performance. In fact, we integrated the presented query answering approach into the reasoning system Konclude and evaluated it with several real-world as well as benchmark ontologies. The comparison with state-of-the-art, but restricted query answering systems shows that our approach often achieves competitive performance or even outperforms these other systems.

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. J. Logic, Language & Information **4**(3) (1995)

[2] PAGOdA ignores the cardinality of answers by evaluating all SPARQL queries as they have the DISTINCT modifier.

3. Calvanese, D., Eiter, T., Ortiz, M.: Answering regular path queries in expressive description logics: An automata-theoretic approach. In: Proc. National Conf. on Artificial Intelligence (2007)
4. Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: HermiT: An OWL 2 reasoner. J. Automated Reasoning **53**(3), 1–25 (2014)
5. Glimm, B., Horrocks, I., Sattler, U.: Unions of conjunctive queries in SHOQ. In: Proc. Int. Conf. on Principles of Knowledge Representation and Reasoning (2008)
6. Glimm, B., Kazakov, Y., Kollia, I., Stamou, G.: Lower and upper bounds for SPARQL queries over OWL ontologies. In: Proc. National Conf. on Artificial Intelligence (2015)
7. Glimm, B., Lutz, C., Horrocks, I., Sattler, U.: Conjunctive query answering for the description logic SHIQ. J. Artificial Intelligence Research **31**, 157–204 (2008)
8. Grädel, E.: Why are modal logics so robustly decidable? In: Current Trends in Theoretical Computer Science, Entering the 21th Century, vol. 2, pp. 393–408. World Scientific (2001)
9. 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)
10. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible SROIQ. In: Proc. Int. Conf. on Principles of Knowledge Representation and Reasoning. AAAI Press (2006)
11. Horrocks, I., Tessaris, S.: Querying the semantic web: a formal approach. In: Proc. Int. Semantic Web Conf. Springer (2002)
12. Hudek, A.K., Weddell, G.E.: Binary absorption in tableaux-based reasoning for description logics. In: Proc. Int. Workshop on Description Logics. vol. 189. CEUR (2006)
13. Kollia, I., Glimm, B.: Optimizing SPARQL query answering over OWL ontologies. J. Artificial Intelligence Research **48**, 253–303 (2013)
14. Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y., Liu, S.: Towards a complete OWL ontology benchmark. In: Proc. European Semantic Web Conf. Springer (2006)
15. Ortiz, M., Calvanese, D., Eiter, T.: Data complexity of query answering in expressive description logics via tableaux. J. Automated Reasoning **41**(1), 61–98 (2008)
16. Pan, J.Z., Thomas, E., Zhao, Y.: Completeness guaranteed approximation for OWL-DL query answering. In: Proc. Int. Workshop on Description Logics. vol. 477. CEUR (2009)
17. Parsia, B., Matentzoglu, N., Gonçalves, R.S., Glimm, B., Steigmiller, A.: The OWL reasoner evaluation (ORE) 2015 competition report. J. Automated Reasoning **59**(4), 455–482 (2017)
18. Rudolph, S., Glimm, B.: Nominals, inverses, counting, and conjunctive queries or: Why infinity is your friend! J. Artificial Intelligence Research **39**, 429–481 (2010)
19. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. J. Web Semantics **5**(2), 51–53 (2007)
20. 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-ISWC.pdf
21. Steigmiller, A., Glimm, B., Liebig, T.: Reasoning with nominal schemas through absorption. J. Automated Reasoning **53**(4), 351–405 (2014)
22. Steigmiller, A., Liebig, T., Glimm, B.: Konclude: system description. J. Web Semantics **27**(1) (2014)
23. Stoilos, G., Stamou, G.: Hybrid query answering over OWL ontologies. In: Proc. European Conf. on Artificial Intelligence (2014)
24. Vardi, M.Y.: Why is modal logic so robustly decidable? In: Proc. DIMACS Workshop on Descriptive Complexity and Finite Models. vol. 31. American Mathematical Society (1997)
25. Zhou, Y., Cuenca Grau, B., Nenov, Y., Kaminski, M., Horrocks, I.: PAGOdA: Pay-as-you-go ontology query answering using a datalog reasoner. J. Artificial Intelligence Research **54**, 309–367 (2015)