Absorption-Based Query Answering for Expressive Description Logics – Technical Report

Andreas Steigmiller and Birte Glimm

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

Abstract. An important reasoning task for logic-based knowledge representation formalisms, such as Description Logics, is conjunctive query answering, where one can 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 for the queries or 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 detected in the model construction process with minor extensions of tableau calculi, which are dominantly used for reasoning with more expressive Description Logics. We implemented the approach in the reasoning system Konclude and our evaluation shows that it can outperform other query answering systems even for queries that are restricted to the capabilities of these 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 $\mathcal{SROIQ}$ [11] 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, due to the wide range of developed optimisation techniques for tableau-based reasoning systems, they are typically able to handle standard reasoning tasks (e.g., consistency

checking, instance retrieval, classification, etc.) for 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., [3,18,21]), the used techniques are often not directly suitable for practical implementations. More precisely, the used techniques are usually not based on tableau algorithms or it is required that *all* possible models of an ontology are systematically constructed, which easily becomes infeasible in practice. Other approaches try to syntactically match the abstractions of models that are generated by model construction calculi (such as tableau) to the query, but it is non-trivial to ensure termination and completeness for these approaches [4,15]. For the DLs $\mathcal{SHIQ}$ and $\mathcal{SHOQ}$, approaches have been developed that reduce conjunctive query answering to instance checking (e.g, [6,8,12]), 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 state-of-the-art reasoning systems.

Even for queries with only answer variables (conjunctive instance queries), existing approaches (e.g., [10,14,24]) 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, also new approaches have been developed that can answer some conjunctive queries more efficiently by computing lower and upper bounds for query answers from a model abstraction built by a reasoner [7]. Lower and upper bound approximations of the ontology can further be used to handle parts of the reasoning work with specialised techniques for less expressive DLs [19,31]. 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 [30]. However, the specialised procedures are still used as a black-box and, furthermore, delegating all work to them is not possible in general and, hence, practical conjunctive query answering techniques for expressive DLs are still urgently 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 [27]. The approach is correct and terminates for DLs

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	$\forall R.C$	$\{\delta \mid \langle \delta, \gamma \rangle \in R^{\mathcal{I}} \to \gamma \in C^{\mathcal{I}}\}$
	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	$a \approx b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$

for which decidability of conjunctive query answering 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 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 and corresponding optimisations in the reasoning system Konclude and some first experiments show encouraging results. Our comparison with other (more restricted) query answering techniques further shows that Konclude often outperforms the state-of-the-art for query answering with more expressive DLs.

The paper is organised as follows: Section 2 gives a brief introduction into DLs and reasoning. Section 4 discusses some general difficulties of query answering with more expressive DLs and sketches some considerations towards a practical implementation. Section 4 then describes our absorption-based query entailment checking technique, for which reductions from query answering are sketched in Section 5. Section 6 discusses the implementation and evaluation results.

2 Preliminaries

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, where some axioms are typically considered as part of the TBox (e.g., $C \sqsubseteq D$) and some as part of the ABox (e.g., $C(a)$), i.e., the TBox and the ABox of a $\mathcal{K}$ are certain subsets of $\mathcal{K}$. 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 tlesuple of answer variables, Y the tuple of existential variables (disjoint to X), and each q_i is either a concept term of the form $C(z)$ or a role term of the form $r(z_1, z_2)$ with $z, z_1, z_2 \in \mathsf{vars}(Q(X, Y))$, where $\mathsf{vars}(Q(X, Y))$ is the set of variable names that occur in Q. Note that we omit the tuple of existential variables, e.g., by writing $Q(X)$, if they are clear from the context. A *Boolean query* $Q(\langle\rangle, Y)$, short Q, is a query without answer variables. To simplify the handling with inverse roles, we consider $r(x, y) \in Q$ as equivalent to $r^-(y, x) \in Q$. 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 A 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 total function π that is a match for $\mathcal{I}$ and Q and for which $\pi(x_i) = a_i^\mathcal{I}$ 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 concept terms, i.e., not as constants in query terms, and we assume that all variables are connected via role terms.

2.2 Tableau Algorithm

Tableau algorithms are dominantly used for reasoning with more expressive DLs and they decide the consistency of a knowledge base $\mathcal{K}$ by trying to construct an abstraction of a model for $\mathcal{K}$, a so-called "completion graph". A completion graph G is a tuple $(V, E, \mathcal{L}, \dot{\neq})$, where each node $v \in V$ (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, for which we also use the notation $C(v)$ ($r(v, w)$). We say a node v is a nominal node if $\{a\} \in \mathcal{L}(v)$ and, otherwise, a blockable node.

The algorithm works by initialising the graph with one nominal node for each individual in the input knowledge base and adding concepts and roles to the node and edge labels as specified by concept and role assertions. Complex concepts are then decomposed using a set of expansion rules, where each rule application can add new concepts to node labels and/or new nodes and edges to the completion graph, thereby explicating the structure of a model. 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., both C and $\neg C$ in a node label), 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. The infinite generation of new nodes is prevented with cycle detection techniques such as *pairwise blocking* [11].

For a concept of the form $\leqslant n\, r.C$ in the label of a node v, the tableau algorithm has to ensure that v has at most r-neighbours with C in their label. This is realised by (non-deterministically) choosing C or $\neg C$ for each r-neighbour and then by merging some neighbours (if there are more than n). If v is a nominal node and there exists a blockable r^{-}-predecessor, i.e., an edge to the nominal node was created from a blockable node, then the tableau algorithm has to fix the number of potential neighbour nodes. This is realised with fixed number of new nominal nodes that are added as r-neighbours of v, i.e., nodes with new nominals in their label that do not yet occur in the completion graph. In particular, the blockable node could be caused from a cyclic concept such that identical blockable nodes are repeatedly required. Since these blockable node have to be merged into the new nominal nodes, pairwise blocking cannot prevent an expansion that could result in a clash (e.g., if the number of neighbours for a nominal is limited with an atmost restriction, but a cyclic concept requires an infinite path of certain successors with links to the nominal). As one can see, new nominals may only be required if inverse roles, atmost number restrictions, and nominals are used in certain combinations in the knowledge base. For $\mathcal{SROIQ}$, there exists an upper bound of potentially required new nominals [11], which depends on the maximum length of paths of blockable nodes that can be constructed before they are blocked.

For handling axioms of the form $A \sqsubseteq C$, one typically uses special lazy unfolding rules in the tableau algorithm, which add the concept C to a node label if it contains the atomic concept A. Axioms that cannot directly be handled with these lazy unfolding rules must be internalised, which can be realised by expressing a GCI $C \sqsubseteq D$ by $\top \sqsubseteq \neg C \sqcup D$. Given that $\top$ is satisfied at each node, the disjunction is then also added to all node labels. Since internalisation is quite inefficient, one typically uses a preprocessing step called absorption. Basically, axioms are rewritten into (possibly several) simpler concept inclusion axioms such that lazy unfolding rules in the tableau algorithm can be used and, therefore, internalisation of axioms is often not required, which typically results in less non-determinism. Absorption algorithms based on binary absorption [13] allow for and create axioms of the form $A_1 \sqcap A_2 \sqsubseteq C$, whereby also more complex axioms can be absorbed. This requires the addition of a (binary) unfolding rule that adds C to node labels if A_1 and A_2 are present.

3 Towards Practical Query Answering for Expressive DLs

In the following we focus on query answering techniques that are based on tableau algorithms since they are dominantly used for reasoning with expressive DLs. In particular, reasoning systems based on tableau algorithms have already shown that they can handle standard reasoning tasks for many real-world ontologies quite efficiently and, therefore, they also seem a good starting point for conjunctive query answering. However, since typical implementations of tableau algorithms are already not worst-case optimal for standard reasoning tasks, it can be doubted that they are worst-case optimal w.r.t. query answering. Nevertheless, query answering based on other (or possibly new) procedures may further require the development of optimisations such that real-world ontologies can efficiently be handled. In addition, the implementation of fully fledged reasoning procedures for expressive DLs is usually a lot of effort, which one typically tries to avoid.

Query answering for more expressive DLs is particularly difficult due to several reasons. First of all, more expressive DLs do not have a finite tree-shaped model property, i.e., we have to deal with infinite models and/or exceptions to the tree-shaped structures. In addition, we have to consider many models that can vary significantly due to the non-determinism. Moreover, if complex roles are used in queries, then it gets difficult to restrict the investigation to certain parts of models. Clearly, for the tree-shaped parts of a query, we can do a well-known rolling up, i.e., rewriting the tree-shaped query parts with only existential variables to complex existential restrictions such that the entailment of these parts can be checked with instance tests. For example, in the query $Q(\langle x, y\rangle) = \{r_1(x, y), r_2(x, y), s_1(y, z), A(z), s_2(z, z'), s_3(z, z'')\}$, the existential variables z, z', and z'' can be eliminated together with the related query terms by "rolling them up" into the existential restriction $\exists s_1.(A \sqcap \exists s_2.\top \sqcap \exists s_3.\top)$ such that only the query terms $r_1(x, y)$, $r_2(x, y)$, and $\exists s_1.(A \sqcap \exists s_2.\top \sqcap \exists s_3.\top)(y)$ remain. Much more problematic are cyclic parts and determining whether they map to some existentially restricted elements in the model, which we consider in the following in more detail. For this, we first focus on Boolean queries, where the tree-shaped parts are rolled up such that only cyclic parts remain, and we first consider them for knowledge bases with basic language features of expressive DL (e.g., $\mathcal{ALCH}$).

In the model construction process, i.e., in the partially built completion graphs, we directly want to check whether the cyclic parts could be satisfied, i.e., for some tree-shaped structure of constructed nodes. These checks can eb realised by "folding" the relational structure of (parts of) the query into the tree-shaped forms of completion graphs 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 $Q_1 = \{A(w), r(w, x), B(x), r^-(x, y), C(y), r(y, z), D(z), r^-(z, w)\}$ (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 $A \sqcap \exists r.(B \sqcap D \sqcap \exists r^-.C)$ and $B \sqcap \exists r^-.(A \sqcap C \sqcap \exists r.D)$, respectively.

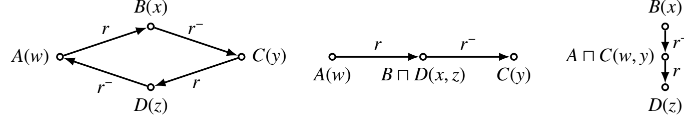

Fig. 1. Visualisation of the query of Example 1 and two possible foldings that are obtained by identifying variables

Evaluating whether a concept representing a folding is satisfied by some (blockable) nodes in the completion graph can be checked deterministically, i.e., if new concepts or roles are added to nodes or edge labels, then we check whether the new concepts or roles lead to the satisfaction of outer (sub-)concepts that represent foldings based on the already satisfied (sub-)concepts of foldings for a node. For instance, if we have a node v with the concept C in its label, then clearly the sub-concept C of the folding $A \sqcap \exists r.(B \sqcap D \sqcap \exists r.C)$ is satisfied for v. If the tableau algorithm creates an r^--successor node v' for v through an application of a rule, then we can deterministically detect that now the outer sub-concept $\exists r.C$ of the first folding is satisfied for v'. If we store/remember those sub-concepts of foldings that are satisfied for a node, then we can incrementally check whether corresponding outer sub-concepts are satisfied after new rule applications. To make these checks even more efficient, one can further "index" the concepts of foldings and/or use a rule engine that is fed with each addition of a concept or role fact. Of course, to enable such a deterministic evaluation of satisfied (parts of) foldings, we have to ensure that the tableau algorithm is indeed deciding whether concepts that occur in the query are satisfied for a node. This can be realised by forcing the addition of disjunctions of the form $\neg C \sqcup C$ to each node label (e.g., by adding an axiom of the form $\top \sqsubseteq \neg C \sqcup C$) if C is a concept term in the query. This clearly does not have any semantic impact, but it simplifies the evaluation of the satisfied foldings. In practice, one would obviously try to add this decisions only to those node labels for which they are relevant, e.g., we would add $\neg B \sqcup B$ and $\neg D \sqcup D$ for node v' in the example above.

We can also add recognized sub-concepts of foldings to the corresponding node labels without interfering the tableau algorithm since the added concepts are already satisfied and so, in principle, they do not have to be processed. Even if the tableau algorithm were to process them, it would not lead to significant changes since they encode only structures that are already present in the completion graph. However, the addition of satisfied sub-concepts of foldings to node labels is useful for ensuring that blocking is working correctly. Since pairwise blocking is typically used for more expressive DLs, we compare, among others, whether the node labels are the same, i.e., the expansion of nodes is only blocked if the same sub-concepts of foldings are present in node labels and, therefore, we ensure that the same query parts are entailed in an analogous way. Roughly speaking, if we found a fully expanded and clash-free completion graph that does not satisfy an entire folding, then we can guarantee that an unravelling of the blocking part is possible without entailing the corresponding query. In fact, if we were to repeat the part between the blocker node and the blocked node repeatedly at the position of the blocked node, then not more (sub-)concepts of the foldings are satisfied since we only repeat the existing structure. In other words, if a new (sub-)concept of

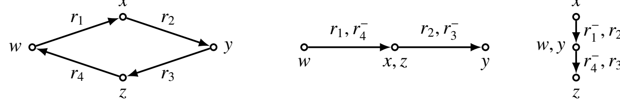

Fig. 2. Visualisation of the query of Example 2 and two possible foldings

a folding were satisfiable due to repetitions, then it would already have been the case for the node structure between the blocker and the blocked node. In particular, checking whether new (sub-)concepts of foldings are satisfied can be done locally by only considering the label of the current node and the roles in edge labels to neighbours. Since all of these local restrictions are checked by the pairwise blocking condition, new (sub-)concepts of foldings cannot be satisfied through an unravelling process. Termination is obviously still guaranteed since we have only a limited number of concepts from the folding process.

Of course, there can be exponentially many foldings and, in general, we also need role conjunction to express them in form of existential restrictions.

Example 2. For the Boolean query $Q_2 = \{r_1(w, x), r_2(x, y), r_3(y, z), r_4(z, w)\}$ (cf. Figure 2), we can only express the foldings in form of DL concepts by using role conjunctions, e.g., with the concepts $\exists(r_1 \sqcap r_4^-).\exists(r_2 \sqcap r_3^-).\top$ and $\exists(r_1^- \sqcap r_2).\exists(r_4^- \sqcap r_3).\top$.

However, if we evaluate the foldings as described above, we do not really require a tableau algorithm that supports such role conjunction since corresponding existential restrictions are only added to node labels for blocking if they are satisfied, i.e., the tableau algorithm can simply ignore them.

In case the knowledge base contains nominals, then we get exceptions to the tree structures. Although even with nominals/individuals, forest-shaped models exists [21], many existing tableau-based reasoning systems built completion graphs that may contain "real cycles". But since we know that these cycles include these nominals, we can simply also use them to built appropriate "foldings" of queries. In particular, a cyclic query where one variable could match to a nominal/individual can be considered with concept expressions, for which the variable is replaced with the nominal and the remaining parts of the query are folded/rolled up. For example, one concept that expresses the query $\{r(x, y), s(y, z), t(z, x)\}$ is $\exists r.\exists s.\{a\} \sqcap \exists t^-.\{a\}$, for which we replaced the variable z with the nominal $\{a\}$. If we do this kind of "folding" for all variables in the cycle with all nominals in the knowledge base (additionally to the ordinary folding with identifying variables), then we can again reuse the previously described evaluation of satisfied concepts of foldings in completion graphs for detecting query entailment. In particular, the pairwise blocking ensures again that an unravelling is possible. If the blocker as well as the blocked node contains the same sub-concepts of foldings (e.g., $\exists t^-.\{a\}$), then repetitions have the same links to the nominal nodes and it is not suddenly possible that other sub-concepts of foldings are satisfied (e.g., $\exists s.\{a\}$).

In principle, other language features of more expressive DLs also require some extensions to the (creation process of) foldings. If a transitive role is used in a role term of the query, then we have to consider cases where this role term can be folded several times with other role terms. For example, the query $\{r(x, y), s(y, z), t(z, x)\}$ cannot

be folded into a tree-shaped concept expression if all roles are only simple (and we do not have self loops in the knowledge, e.g., due to concepts of the form $\exists r'.\mathsf{Self}$). But if, for example, the role t is transitive in the corresponding knowledge base, then the role term for t can be mapped to an arbitrary long path of edges that are labelled with t in the completion graph. Hence, we may have to reuse such role terms several times for the creation of foldings. As a consequence, we get for this query the foldings $\exists(r \sqcap t^-).\exists(s \sqcap t^-).\top$ and $\exists(r^- \sqcap s).\exists(t \sqcap t^-).\top$. In principle, also $\exists(r^- \sqcap s).\exists(t \sqcap t^-).\exists(t \sqcap t^-).\top$ would be a valid folding, but it is sufficient to consider foldings that are not subsumed by others and $\exists(r^- \sqcap s).\exists(t \sqcap t^-).\exists(t \sqcap t^-).\top$ is obviously subsumed by $\exists(r^- \sqcap s).\exists(t \sqcap t^-).\top$.

A systematic way of creating foldings in presence of transitive roles is by having a separate step that transforms a role term with the transitive role, say $t(x, y)$, to a concept term of the form $\exists(t^- \sqcap t).\top(x)$ (and by replacing y with x in the remaining query) or that splits it into two role terms with the introduction of an intermediate variable, e.g., $t(x, y') \wedge t(y', y)$. In the latter case, we then have to identify the intermediate variable (y') with a neighbour variable of x or y (e.g., z if we additionally have the role term $s(z, x)$) or, alternatively, we replace the intermediate variable with a nominal of the knowledge base. If we do this step in addition to the ordinary folding step, where we simply identify a variable with another variable (that is not a direct neighbour), and the ordinary nominal replacing step (with the subsequent rolling up) such that all possible combinations of steps are applied in all ways, then we basically get the foldings for $\mathcal{SHOIQ}$ knowledge bases. Of course, the above described query entailment checking method in the constructed completion graphs would be incomplete since new nominals are not considered (which may be introduced by the tableau algorithm to fix the number of neighbours for individuals or other nominals). Clearly, one could also create foldings with those new nominal names that will be used by the tableau algorithm, but since the maximum number of new nominals is limited by the blocking technique and the detection of concepts that represent foldings influences the blocking, we cannot limit the number of new nominals upfront and, hence, completeness or termination (if the foldings for new nominals are dynamically created) cannot be guaranteed.

Other features of $\mathcal{SROIQ}$ are practically also not completely trivial. In particular, complex roles (that are not only transitive) can cause much more foldings. If we have, for example, the role inclusion axiom $r_1 \circ r_2 \circ s \sqsubseteq s$, then we can encode in the foldings how often the role s is "expanded" via this axiom. In fact, if we have a query consisting of only the role term $s(x, x)$ and a knowledge base with the nominal a, then one possible folding is $\exists s^-.\{a\} \sqcap \exists r_1.\exists r_2.\{a\}$, whereas another valid folding is $\exists s^-.\{a\} \sqcap \exists r_1.\exists r_2.\exists r_1.\exists r_2.\{a\}$, and, in principle, we can continue the "expansion" of s arbitrarily to get more foldings. A direct limit of the expansion is non-trivial since the described paths could not only include all individuals, but also an unlimited number of anonymous elements, which again seems problematic for blocking. However, in this case we can regain a finite number of folding/query concepts by introducing some auxiliary axioms of the form $r_1 \circ r_2 \sqsubseteq t'$ and $t' \circ t' \sqsubseteq t'$ with the new (transitive) role t' such that the concept $\exists s^-.\{a\} \sqcap \exists t'.\{a\}$ compactly encodes the possible expansions described above. Although it is also clear that the number of foldings is limited if we check for an intersections of several complex roles, e.g., with a query of the form

$\{s_1(x, y), s_2(x, y), \ldots, s_n(x, y)\}$ for $s_1, s_2, ..., s_n$ complex roles, it can be non-trivial to create all these foldings systematically.

Since the DL $\mathcal{SROIQ}$ further allows for expressing direct loops for an element, e.g., with self concepts of the form $\exists r.\mathsf{Self}$ or with axioms that state that a (possibly complex) role is (globally) reflexive, we have to consider the possibility of these loops in the foldings. For this, we can add a step to the folding process that replaces a role $r(x, y)$ with a concept term of the form $\exists r.\mathsf{Self}(x)$ and the variable y with x for the remaining query if the knowledge base contains axioms that may lead to the creation of such direct r-loops in the completion graph. Of course, we also have to adapt the checks in completion graphs to recognize these loops such that the (sub-)concepts of obtained foldings can be added as soon as they are satisfied.

As discussed, if we ignore new nominals, then all possible "foldings" could be created upfront and checking whether (or which (sub-concepts) of) foldings are satisfied is straightforward in the completion graph. But since we can get exponentially many foldings and their construction process can be non-trivial (especially with complex roles), we are not really interested in actually making them explicit. This can be addressed, at least to some extent, by *dynamically* creating them as they occur in the completion graphs, i.e., if we have a query $\{s(x, y), r(x, y)\}$ and we detect that a nominal node has a new s-neighbour, then we check whether this could be extended to a folding (e.g., $\exists s^-.\{a\} \sqcap \exists r.\{a\}$) and if this is the case, then we add the corresponding (sub-)concepts to the nodes (e.g., $\exists s^-.\{a\}$ to the label of the s-neighbour). Clearly, this requires a sophisticated indexing of the query and an extended management of potential foldings (or how we can get to them), which seems non-trivial especially for arbitrary complex roles. Although we can still get exponentially many concepts from foldings in the worst-case, we can expect that much less concepts are actually needed/created for real-world ontologies since we only collect those that actually occur in the constructed completion graphs. In the next section, we present a special encoding of satisfied (sub-)concepts of foldings such that the number of concepts is further reduced and for which we do not require a (sophisticated) management of possible foldings.

Another problem of the above described dynamic detection method of possible foldings is that it does not directly work for the ABox part of the knowledge base. In particular, individuals can be arbitrarily connected and, as a consequence, they can form real cycles to which foldings cannot be matched. Of course, we can treat all individuals as nominals such that they are considered in the foldings, but that would increase the number of (possible) foldings quite dramatically and also the cost of their management. In addition, we have to be careful with the creation of foldings such that the same query terms are not evaluated several times, as demonstrated with the following example.

Example 3. Let us consider the query $\{r(x, y), s(y, z), t(z, x)\}$ and a knowledge base with the individuals a, b, and c, which are connected via the role assertions $r(a, b)$, $s(b, c)$. If we naively create the foldings, i.e., by simply replacing one variable in the cycle with an individual treated as nominal, then we get, among others, the concepts $\exists r.\exists s.\{c\} \sqcap \exists t^-.\{c\}$ (by replacing z with $\{c\}$) and $\exists s.\exists t.\{a\} \sqcap \exists r^-.\{a\}$ (by replacing x with $\{a\}$) for foldings although we do the rolling up of the remaining part of the query in the same direction. Consequently, we would detect $\exists r.\exists s.\{c\}$ for the node for x and $\exists r^-.\{a\}$ for the node for

y even if we only dynamically create the foldings, i.e., the role term $r(x, y)$ is basically evaluated twice.

The redundant evaluation happens since we do not know the position of nominals in the query and, therefore, we have to evaluate all possibilities. We may get much more redundant evaluations if the query can go over anonymous elements/blockable nodes, has larger cycles, or uses complex roles. Moreover, we are often interested in determining all (relevant) individuals that match the query in order to get candidates for query answering, which would require more adaptations to the folding creation and/or detection process and could further increase the number of foldings. As presented in the next section, our special encoding of detected foldings also addresses these problems.

If we are able to create all concepts of foldings of queries upfront and if we have a tableau algorithm that can process (restricted occurrences of) role conjunctions, then we can directly encode the query entailment problem into the knowledge base with axioms of the form $C_f \sqsubseteq \bot$, where C_f is a concept representing a folding. Clearly, this does not consider new nominals, i.e., this procedure is incomplete for $\mathcal{SROIQ}$, but it allows the reasoner to use typical preprocessing steps for optimising reasoning (e.g., absorption). On the other hand, if the reasoner does not (fully) support incremental reasoning, then the reasoner may be force to retest the consistency of the entire knowledge base for each query/query entailment check. Nevertheless, the number of possible foldings can get quite large, therefore, this naive reduction to many axioms does not seem to be a good idea in the general case. However, we show in the next section that we incorporate this idea by creating a few special axioms for a query, which can then further be preprocessed by the reasoner and we use special encodings based on variable bindings for the representation of possibly many foldings in the completion graph.

4 Absorption-Based Query Entailment Checking

Instead of directly collecting all possible foldings of queries in form of concepts in the corresponding node labels, we rewrite the query into several simple axioms that they can efficiently be processed with minor extensions of the tableau algorithm. For correctly capturing the semantics of the query with split up axioms, we use constructs that force the tableau algorithm to collect bindings for variables, which are then propagated through the completion graph accordingly to the query with correspondingly generated concepts and axioms. The occurrence of a generated concept in a node label then basically indicates the state of the query, i.e., how much of the query is satisfied, whereas the gathered bindings for variables indicate how the query is satisfied, i.e., how the satisfied part of the query is folded. For example, we obtain from the Boolean query $Q_3 = \{t(w, x), r(x, y), s(y, z), s(z, w)\}$ the concept $\exists(t \sqcap s^-).\exists(r \sqcap s^-).\top$ for the folding where we identify z and x (cf. Figure 3). By creating bindings for the variable w to nodes in completion graphs and by propagating these bindings w.r.t. the role terms of the query (i.e., first to t-neighbours, then successively to r, s, and s-neighbours), we can detect whether a binding is propagated back to the node from which it stems and, thus, whether the graph contains a cycle such that the query is entailed. If we additionally create and propagate bindings for x, y, z (which is, in principal, not necessary for

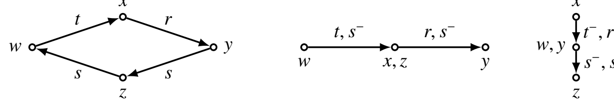

Fig. 3. Boolean query $Q_3 = \{t(w, x), r(x, y), s(y, z), s(z, w)\}$ with two possible foldings that correspond to the concepts $\exists(t \sqcap s^-).\exists(r \sqcap s^-).\top$ and $\exists(t^- \sqcap r).\exists(s^- \sqcap s).\top$

simply checking entailment of the query), then we can even identify which foldings are satisfied (e.g., the folding for the concept $\exists(t \sqcap s^-).\exists(r \sqcap s^-).\top$ is satisfied if x and z are bound to the same node). For the propagation of bindings, we utilise concepts of the form $\forall r.C$ (for which the $\forall$-rule has to be extended/adapted appropriately), whereas the creation of variable bindings is realised with $\downarrow$ binders borrowed from Hybrid Logics [2], i.e., concepts of the form $\downarrow w.C$. The idea of the $\downarrow$ binders is basically to store the elements (of the model) for the given variable such that they can be referred in the following (e.g., in the sub-concept). Consequently, a binder concept $\downarrow w.C$ can be seen as an instruction for the tableau algorithm to create a binding for w to the node that has $\downarrow w.C$ in its label and to store the binding for the sub-concept C. To combine these different concepts and to incorporate disjunctions for concept terms, we use (binary) inclusion axioms of the form $S \sqcap A \sqsubseteq C$ (with appropriately adapted unfolding rules).

Since the query rewriting process reminds of absorption and our approach can be seen as an extension of handling rule-based knowledge in ontologies encoded via nominal schemas [27], we also refer to the rewriting process as query absorption. In particular, it also reduces non-determinism by avoiding/delaying decisions of the form $\neg C \sqcup C$ for a concept term $C(x)$ until the preceding part of the query is satisfied for a node (instead of forcing the decision for all nodes, e.g., with axioms of the form $\top \sqsubseteq \neg C \sqcup C$). We next present a basic query absorption algorithm in Section 4.1. Subsequently, we discuss the required extensions of the tableau algorithm in Section 4.2. Beside extending the expansion rules such that bindings are created and propagated, we also have to adapt the blocking check slightly since we are no longer representing all discovered foldings directly. Finally, in Section 4.3 we discuss a few straightforward (absorption) optimisations. Please note that we focus in this section on query entailment checking and show (optimised) reduction techniques from query answering in Section 5.

4.1 Query Absorption

In principle, our query absorption approach also utilises that entailment checking of a query with the terms $\{q_1, \ldots, q_n\}$ can be reduced to consistency testing by adding a rule/axiom of the form $q_1 \wedge \ldots \wedge q_n \sqsubseteq \bot$ to the knowledge base. Since the left-hand side, however, cannot directly be checked by the tableau algorithm, we split it up into single steps for which we then have to encode the intermediate state of the query. We do this in form of *query state concepts*, which are basically used analogously to fresh atomic concepts, but in order to correctly/fully interpret them, we have to consider the associated bindings of variables. In fact, the query state concept $S^{q_1,q_2,\ldots,q_n}$ can be seen as a fresh atomic concept S that signalizes that the query terms $q_1, \ldots, q_n$ are satisfied

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

Input: A conjunctive 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 $q = r(x, y)$ with $z \neq x$ **then**
7: choose $q_1 \cdot q_2 \cdot ... \cdot q_n \in Q$ with $q_1 = r_1(z, y_1), q_2 = r_2(y_1, y_2), ..., q_n = r_n(y_{n-1}, x)$
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: $S^{z_1 ... z_m z} \leftarrow V_{LS}(z)$
22: $\mathcal{K} \leftarrow \mathcal{K} \cup \{S^{z_1 ... z_m z} \sqsubseteq \bot\}$

with the given/associated bindings for the variables. Since the query terms are often clear from the absorption process, we usually only write the query state concepts with the incorporated variables, e.g., $S^{x_1 ... x_n}$, and use sub-scripts to indicate over which roles has been propagated, e.g., S^x_t indicates that bindings for x have been propagated via t.

The absorption of a query Q w.r.t. a knowledge base $\mathcal{K}$ is realised via the function absorbQ shown in Algorithm 1, which extends $\mathcal{K}$ via side effects. The functions absorbCT (see Algorithm 2) and absorbRT (Algorithm 3) are used to appropriately handle concept and role terms by adding corresponding axioms to $\mathcal{K}$. Both 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. The absorption, i.e., the rewriting of the query into split up axioms, now basically works as follows:

- In the beginning, one variable is chosen as the position for which we start the propagation of the query (see Line 1 of Algorithm 1). This variable is then considered as the current (position) variable and the query terms are absorbed relatively to it. For role terms, we update the current variable to the one to which has been propagated. Roughly speaking, the current variable allows for generating an easy to handle sequence of query state concepts, where each following query state concept indicates that more of the query is satisfied than for the preceding one.
- In principle, the algorithm creates for each variable in the query, say z, a binder concept $\downarrow z.S^z$ such that the tableau algorithm creates a binding for z and associates

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

Input: A concept term ($C(x)$), a mapping of variables to the last query state concepts (V_{LS}), and a knowledge base ($\mathcal{K}$) that is extended via side effects

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

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

Input: A role term ($r(x, y)$), a mapping of variables to the last query state concepts (V_{LS}), and a knowledge base ($\mathcal{K}$) that is extended via side effects

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

it with the following query state concept S^z. For the current variable that is chosen at the beginning, we force the addition of the corresponding binder concept to all node labels with an axiom of the form $\top \sqsubseteq \downarrow z.S^z$ (see Line 3 of Algorithm 1). For other variables, we create the binder concepts on demand, i.e., we imply a binder concept for a variable y if we have our first propagation to the position of the variable y (Line 6 of Algorithm 3). Note that the query state concepts are stored for the handled variable with the mapping V_{LS} (see Line 4 of Algorithm 1 and Line 7 of Algorithm 3). Although the binder concepts have already been considered for realising query answering for DLs (see [5]), the presented approach differs in the way how they are used. In fact, we associate the created bindings with query state concepts (instead of directly using the variables in sub-concepts) such that we can split up the axioms to enable a more efficient processing.

- If we have to "absorb" a concept term $C(x)$, then we build a decision of the form $\neg C \sqcup F^x_C$, where F^x_C is a fresh atomic concept (see Line 4 of Algorithm 2). Roughly speaking, we stop the propagation/continuation of query states (and the associated variable bindings) if the concept term is not satisfied for a node, i.e., the node is an instance of $\neg C$. If the concept is satisfied (or it can be assumed that it is satisfiable), then we force the addition of F^x_C, which is then used to continue the query state propagation. Note that we do not have to add C since if the tableau algorithm detects

inconsistency, e.g., due to the entailment of the query, then it is backtracking and, hence, automatically testing whether the alternative with $\neg C$ can be fully expanded to clash-free completion graph. We even want to avoid the addition of C since it could be a complex concept for which a lot of rule applications could become necessary. Note that we do not directly imply a new query state concept, but we use a binary inclusion axiom of the form $S^{x_1 \ldots x_n x} \sqcap F_C^x \sqsubseteq S_C^{x_1 \ldots x_n x}$ (Line 5) such that $S_C^{x_1 \ldots x_n x}$ is implied if $S^{x_1 \ldots x_n x}$ and F_C^x occur in a node label. This is necessary since we do not want to modify the $\sqcup$-rule of the tableau algorithm. In fact, if we were to build an axiom of the form $S^{x_1 \ldots x_n x} \sqsubseteq \neg C \sqcup S_C^{x_1 \ldots x_n x}$ and the tableau algorithm were to chose the disjunct $S_C^{x_1 \ldots x_n x}$, then associated bindings would have to be propagated to the query state concept within the $\sqcup$-rule. By using the query state concepts only in certain axioms and by using them only in combination with certain concept constructors, we can significantly limit the required modifications to the tableau algorithm.
- In contrast, if we want to "absorb" a role term $r(x, y)$, then we build a propagation over the corresponding role via a universal restriction of the form $\forall r.S_r^{x_1 \ldots x_n x}$ (see Line 3 of Algorithm 3). In addition, we use a binary inclusion axiom of the form $S_r^{x_1 \ldots x_n x} \sqcap S^{y_1 \ldots y_m y} \sqsubseteq S^{z_1 \ldots z_k}$ to join the newly created propagation over the role ($S_r^{x_1 \ldots x_n x}$) with the existing propagation encoded in the query state concept $S^{y_1 \ldots y_m y}$ (Line 11). Note that this binary inclusion axiom has query state concepts for both conjuncts on the left-hand side and it has to combine/join the bindings for variables appropriately. Consequently, the new query state concept on the right-hand side includes the variables/query terms of both conjuncts (cf. Line 10).
- Based on the current variable, the query absorption algorithm successively handles/absorbs all query terms, which results in a sequence of query state concepts. If the next chosen query term is at the position of the current variable, then we can directly absorb it, i.e., if the term is a concept term, then we call absorbCT (Line 13) and, otherwise, we call absorbRT (Line 17). Note that after calling absorbRT, we update the current variable to the variable to which has been propagated, i.e., the other variable of the role term (Line 18). In contrast, if the next query term is not at the current position, i.e., it does not contain the current variable, then we first have to go/propagate to the new position (Lines 6–11). This is simply possible by choosing a chain of adjacent role terms (i.e., containing the same variable) from the query such that the first role terms contains the current variable and the last one the variable of the chosen query term (Line 7) and, then, we simply create propagations over this chain by calling absorbRT for each role term in the chain (Line 9).
- In the end, i.e., after all query terms are processed, we take the last query state concept from the current variable and imply $\bot$ (Lines 21–22). This forces the tableau algorithm to do backtracking if that state of the query is reached in the completion graph. Hence, we enforce the search for a model with non-entailment. If such a model cannot be found, then the knowledge base (extended with the axiom generated by the absorption) is reported as inconsistent and we know that the query is entailed.

Clearly, the presented absorption algorithm is far from optimal, i.e., it may create some unnecessary bindings and propagations. In particular, the number of axioms de-

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

Fig. 4. The axioms for absorbing the query Q_3 of Example 4

pends on the initially chosen variable, the order in which the query terms are processed, and the paths we may have to choose from the query to get to the new position of the next query term. In practice, one should obviously try to make these decisions such that most query terms can directly be absorbed without creating additional propagations. However, one should also consider how much effort these propagations can cause in the completion graphs. In fact, if we gather statistics about the instances of concepts and roles in completion graphs (e.g., during consistency testing), then we can use them to first absorb those terms that (probably) have fewer instantiations. Especially complex roles can (indirectly) connect many nodes in completion graphs and, hence, can cause a lot of propagation work, i.e., we are usually interested in delaying the absorption of corresponding role terms. We discuss more absorption optimisations that are useful in practice in Section 4.3, but the presented algorithm is already quit convenient to show the principle of the approach.

Example 4. Query $Q_3 = \{t(w, x), r(x, y), s(y, z), s(z, w)\}$ (cf. Figure 3) can be absorbed with Algorithm 1 with the axioms depicted in Figure 4. The algorithm chooses w as the initial current variable for which then the first axiom is added that implies the binder concept ${\downarrow}w.S^w$ from $\top$ to start the propagation for the query (cf. Lines 1–3 of Algorithm 1). Subsequently, the algorithm processes all the query terms. Since it is convenient to continue with the absorption of those query terms that are at the position of the current variable, we first choose $t(w, x)$, for which then the absorbRT function is called (cf. Line 17) and the next three axioms are created. In particular, first the propagation over the corresponding role is created ($S^w \sqsubseteq \forall t.S^w_t$, cf. Line 3 of Algorithm 3). Since bindings for the variable x have not yet been created, we additionally have to imply a corresponding binder concept ($S^w_t \sqsubseteq {\downarrow}x.S^x$, cf. Line 6) and, then, we have to join the new bindings with the existing propagation ($S^w_t \sqcap S^x \sqsubseteq S^{wx}$, cf. Line 11). Now, x is set as the current variable and we can continue with the next query term. If we now choose $r(x, y)$ and after that $s(y, z)$, then Axioms 4–9 are generated analogously. Subsequently, only the term $s(z, w)$ remains for which again the absorbRT function is called, but since we already have bindings for w, the algorithm does not create a new binder concept. In fact, we use a binary inclusion axiom to join the propagation with the bindings of the last query state concept for the variable w ($S^{wxyz} \sqcap S^w \sqsubseteq S^{wxyzw}$, cf. Line 11). Finally, the algorithm implies $\bot$ from the last query state concept of the current variable ($S^{wxyzw} \sqsubseteq \bot$, cf. Line 22).

As one can observe from the example, the absorption algorithm creates bindings for all variables although not all of them are actually required for joins. If these variables

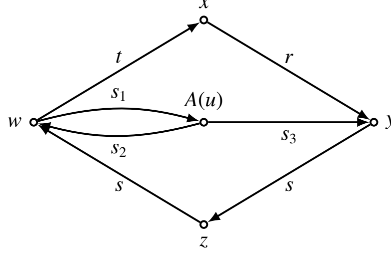

Fig. 5. Boolean query $Q_4 = \{t(w, x),\ r(x, y),\ s(y, z),\ s(z, w),\ s_1(w, u), A(u), s_2(u, w), s_3(u, y)\}$

are, however, answer variables, then we may, nevertheless, be interested in these bindings in order to get answer candidates. In addition, bindings for several variables may be required if the query has more than one trivial cycle.

Example 5. If we assume that $Q_4 = Q_3 \cup \{s_1(w, u), A(u), s_2(u, w), s_3(u, y)\}$ (cf. Figure 5), then, the absorption algorithm can generate, in addition to the Axioms 1–12 from Example 4 (i.e., all except the last one), the following axioms:

$$\begin{aligned} S^{wxyzw} &\sqsubseteq \forall s_1.S^{wxyzw}_{s_1} & S^{wxyzw}_{s_1} &\sqsubseteq {\downarrow} x.S^{u} \\ S^{wxyzw}_{s_1} \sqcap S^{u} &\sqsubseteq S^{wxyzwu} & S^{wxyzwu} &\sqsubseteq \neg A \sqcup F^{u}_{A} \\ S^{wxyzwu} \sqcap F^{u}_{A} &\sqsubseteq S^{wxyzwu}_{A} & S^{wxyzwu}_{A} &\sqsubseteq \forall s_3.S^{wxyzwu}_{s_3} \\ S^{wxyzwu}_{s_3} \sqcap S^{wxy} &\sqsubseteq S^{wxyzwuy} & S^{wxyzwuy} &\sqsubseteq \forall s_3^{-}.S^{wxyzwuy}_{s_3^{-}} \\ S^{wxyzwuy}_{s_3^{-}} \sqcap S^{wxyzwu}_{A} &\sqsubseteq S^{wxyzwuyu} & S^{wxyzwuyu} &\sqsubseteq \forall s_2^{-}.S^{wxyzwuyu}_{s_2^{-}} \\ S^{wxyzwuyu}_{s_2^{-}} \sqcap S^{wxyzw} &\sqsubseteq S^{wxyzwuyuw} & S^{wxyzwuyuw} &\sqsubseteq \bot \end{aligned}$$

In particular, the last query state concept generated by the absorption algorithm for Q_3 is S^{wxyzw} and it has been associated with the position of variable w. If we continue the absorption with w as the current variable and $s_1(w, u)$ as the next role term that is processed, then we build a propagation over the role $s_1(w, u)$ with a new binder concept for u (cf. Axioms 1–3). Since u is now our current variable, we can directly absorb the related concept term $A(u)$, for which the absorption algorithm creates a decision of the form $\neg A \sqcup F^{u}_{A}$ and uses the binary inclusion axiom $S^{wxyzwu} \sqcap F^{u}_{A} \sqsubseteq S^{wxyzwu}_{A}$ to continue the propagation if A is possibly satisfied. If we assume that the absorption is continued with the role term $s_3(u, y)$, then the axioms $S^{wxyzwu}_{A} \sqsubseteq \forall s_3.S^{wxyzwu}_{s_3}$ and $S^{wxyzwu}_{s_3} \sqcap S^{wxy} \sqsubseteq S^{wxyzwuy}$ are generated. Now, we cannot continue with the absorption of a query term containing the current variable, so we first have to build a propagation to the new position. In fact, the only remaining query term is $s_2(u, w)$ and the simplest way to one variable of this query term is by propagating back over the role term $s_3(u, y)$. Hence, we assume that the absorption algorithm (cf. Line 7 of Algorithm 1) picks the chain/path consisting of only the role term $s_3^{-}(y, u)$ and generates a propagation back to the position of variable u via the axiom $S^{wxyzwuy} \sqsubseteq \forall s_3^{-}.S^{wxyzwuy}_{s_3^{-}}$. After joining with the last query state concept for u, we can absorb the last remaining role term $s_2(u, w)$ by creating a corresponding propagation. Finally, we join the propagation with last query state concept for w again, which is then used to imply $\bot$.

4.2 Tableau Rules and Blocking Extensions

A few minor extensions and adaptations to the tableau algorithm are required to correctly handle the axioms that are generated with the query absorption algorithm. In fact, the absorption algorithm splits the query into several smaller axioms that can be processed more easily, but in order to correctly capture/retain the semantics of the query, we have to create and manage some additional bindings for variables. To be more precise, the introduced binder concepts specify where and which bindings for variables have to be created in the completion graph and the remaining concepts and axioms describe how these bindings have to be propagated and joined. If this is realised with corresponding extensions to expansion rules, then the occurrences of (query state) concepts in node labels encode (together with the propagated bindings) possible foldings of the query that are satisfied so far. Since the foldings are only represented indirectly, we further have to extend the blocking conditions such that nodes are only blocked if the query is not entailed even if the blocked nodes are unravelled. In the following, we describe these adaptations to the tableau calculus in more detail.

First, we discuss the required rule extensions to handle bindings for variables in completion graphs. Roughly speaking, we use mappings from variables to nodes for managing these bindings and we store for the concepts in node labels those variable mappings that are associated with them (via the mapping $\mathcal{M}$). In particular, if a binder concept $\downarrow x.C$ occurs in the label of a node v, then we create a (variable) mapping μ with $\mu(x) = v$ in order to "remember" the binding $x \mapsto v$ and we associate the created mapping μ with the concept C for the node v, i.e., $\mathcal{M}(C, v) = \{\mu\}$ (cf. $\downarrow$-rule of Table 2). Note that binder concepts only occur in axioms from the absorption in such a way that we only have to create one new variable mapping with a single binding. Since we can, however, have several propagations, we associate sets of variable mappings with concept facts. More precisely, we define variable mappings (if no confusing is likely to arise, we simply write mapping) and their association with concept facts as follows:

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 $G = (V, E, \mathcal{L}, \neq, \mathcal{M})$*, we denote with* $\mathcal{M}(C, v)$ *the sets of variable mappings that are associated with the concept* C *in* $\mathcal{L}(v) \in V$*.*

If there are mappings associated with concept facts in the completion graph, then we propagate them to other concept facts according to the absorption axioms by using the extensions and adaptations of expansion rules depicted in Table 2. 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, in addition to the behaviour of the original rule that only adds the concept C to the labels of the neighbours. Of course, if complex roles have to be handled, then an unfolding of the universal restriction according to the automata for role inclusion axioms has to be done (see [11]). Alternatively, one can rewrite universal restrictions in the knowledge base such that the unfolding is automatically realised (see, e.g., [22]), which could result in

Table 2. Tableau rule extensions for creating and propagating variable mappings

Rule		Condition / Action
$\downarrow$-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 $\mathcal{M}(\forall r.C, v) \not\subseteq \mathcal{M}(C, w)$
	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 $\mathcal{M}(S^{x_1 \ldots x_n}, v) \not\subseteq \mathcal{M}(C, v)$
	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 $\mathcal{M}(S^{x_1 \ldots x_n}, v) \not\subseteq \mathcal{M}(C, v)$
	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 $(\mathcal{M}(S_1^{x_1 \ldots x_n}, v) \bowtie \mathcal{M}(S_2^{y_1 \ldots y_m}, v)) \not\subseteq \mathcal{M}(C, v)$
	then	$\mathcal{L}(v) = \mathcal{L}(v) \cup \{C\}$ and $\mathcal{M}(C, v) = \mathcal{M}(C, v) \cup (\mathcal{M}(S_1^{x_1 \ldots x_n}, v) \bowtie \mathcal{M}(S_2^{y_1 \ldots y_m}, v))$

an exponential blow up, whereas a dynamic unfolding within the $\forall$-rule may be able to avoid the creation of some propagations if there are no corresponding neighbours.

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, 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 $S^{x_1 \ldots x_n} \sqsubseteq C \in \mathcal{K}$, $S^{x_1 \ldots x_n}$ 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 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 6. 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_3 in Figure 4 and test the consistency with a tableau algorithm extended by the rules of Table 2. We observe that the constructed completion graph

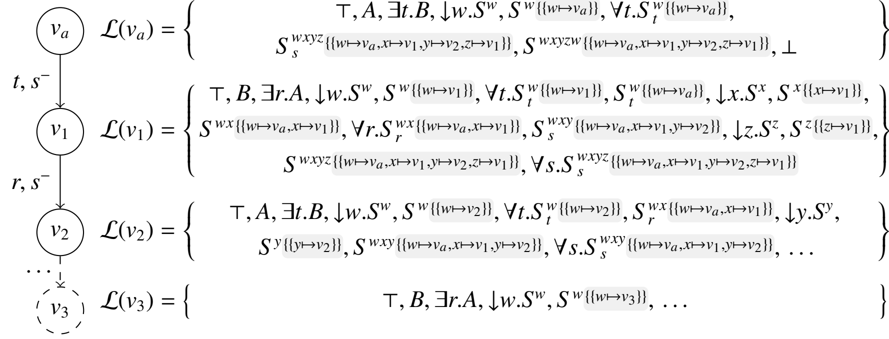

Fig. 6. Completion graph with propagated variable mappings

contains a clash and, consequently, Q_3 is entailed (cf. Figure 6). 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_3 (cf. Figure 3) is satisfied for each node that instantiates A.

Due to $\top \sqsubseteq \downarrow w.S^w$ from the query 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 6, 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 $\mathcal{M}(S^{wxyzw}, v_a)$ and, after one more unfolding with the $\sqsubseteq_1$-rule, we have $\bot \in \mathcal{L}(v_a)$. Since all facts and variable mappings are derived deterministically, no non-deterministic alternatives have to be evaluated and entailment of Q_3 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 rolling up the remaining terms. Note that a variable mapping can represent several foldings, e.g., $\{w \mapsto v', x \mapsto v', y \mapsto v', z \mapsto v'\}$ represents the folding $\exists(t \sqcap s^-).\exists(r \sqcap s^-).\top$ but also $\exists(r \sqcap t^-).\exists(s \sqcap s^-).\top$ of query Q_3 of Example 4. In contrast, we can also have several variable mappings that represent the

same folding, e.g., $\{w \mapsto w_0, x \mapsto w_1, y \mapsto w_2, z \mapsto w_1\}$ and $\{w \mapsto w_0, x \mapsto w'_1, y \mapsto w'_2, z \mapsto w'_1\}$ represent the folding $\exists(t \sqcap s^-).\exists(r \sqcap s^-).\top$, once over the nodes w_0, w_1, w_2 and once over w_0, w'_1, w'_2. However, the latter problem can be reduced (to some extent) by removing bindings for variables that are no longer used after joins.

Without further restricting the approach, we have to create new bindings for every node. Even if we identify conditions that allow for adding the binder concepts only to certain node labels (see, e.g., Section 4.3), then we still can have variable mappings that are propagated arbitrarily far due to complex roles and/or nominals. At first sight, this seems to be problematic for blocking. The correspondence with foldings, however, helps us to find a suitable extension of the typically used pairwise blocking technique [11] 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 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. For this, we can, in principle, simply check (in addition to pairwise blocking) whether each variable mapping μ_v associated with a query state concept $S^{x_1 \ldots x_n}$ in the label of v (i.e., the node that is to be blocked) has a corresponding variable mapping μ_w that is associated with $S^{x_1 \ldots x_n}$ in the label of the potential blocker node w, where $\mu_w(x_i) = \mu_w(x_j)$ if $\mu_v(x_i) = \mu_v(x_j)$ and $\mu_w(x_i) = v_o$ if v_o is nominal node and $\mu_v(x_i) = v_o$. The same must also hold for the predecessors and also in the other direction, i.e., for each variable mapping in v' (w, w'), there also must be a corresponding variable mapping in w' (v, v'). It can easily be seen, that the completion graph construction with such a blocking extension must terminate (if no new nominal nodes are created) since we only have a fixed number of query state concepts and the possible combinations for the variable mappings are limited, too. Also note that an unravelling is guaranteed (if no new nominal nodes are created and no complex roles are used in role terms) since the propagation of query state concepts only takes place over direct neighbours and the (extended) pairwise blocking checks an adjacent pair of nodes, i.e., if the unravelling of a blocked node had implied a new query state concept, then it would already be the case for the part of the completion graph/model that is repeated.

Although such a blocking condition is nicely checkable, it causes some issues in practice. On the one hand, we cannot easily support complex roles with it since the "unfolding/expansion" of universal restrictions does not introduce new binders but only intermediate query state concepts and, hence, propagations over nominals cannot directly be detected. On the other hand, we do not want to create and propagate bindings for all variables (see optimisations discussed in Section 4.3). If we, however, omit bindings for some variables, then we can also not directly check whether the propagation goes over nominals, which is required for the blocking test.

As a remedy, we can use the query state concepts with which the variable mappings are associated to get some indication how the mappings have been propagated. In fact, 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. Hence, even if we do not create new bindings for (nominal) nodes, then we can identify related propagations over these (nominal) nodes by checking whether there are query state concepts associated with compatible variable mappings. Again, we have to ensure that the propagation between the blocked node, its predecessor, and the (related) nominal nodes is the same as/analogous to the propagation between the blocking node, its predecessor, and the (related) nominal nodes. Consequently, we check whether there exist corresponding variable mappings for both group of nodes that have been propagated analogously by comparing the related (query state) concepts, i.e., concepts with which compatible mappings are associated.

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 define states to contain only query state concepts since we also want to capture universal restrictions that are involved in propagating variable mappings (e.g., $\forall t.S_t^w$), which enables more optimisation possibilities (see Section 4.3). Now, we can 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* $1 \leq i \leq n$*.*

Note that it would be sufficient to only consider maximal mappings and with which concepts their sub-mappings are associated if the absorption algorithm only generates one sequence of query state concepts (as the algorithm presented in Section 4.1), i.e., each query state concept indicates that at least as much of the query is satisfied as for the previous query state concept. However, the consideration of compatible variable mappings allows for absorption optimisations that create simultaneous propagations over several role terms (cf. Section 4.3), which can be especially useful for answer variables (e.g., if a small upper bound for an answer variable can be incorporated to restrict the remaining propagation (cf. Section 5.2)). Although the consideration of compatible variable mappings can further be refined (e.g., by considering only compatible variable mappings that also have at least one common binding), we are already able to reliable block an infinite expansion for cyclic concepts.

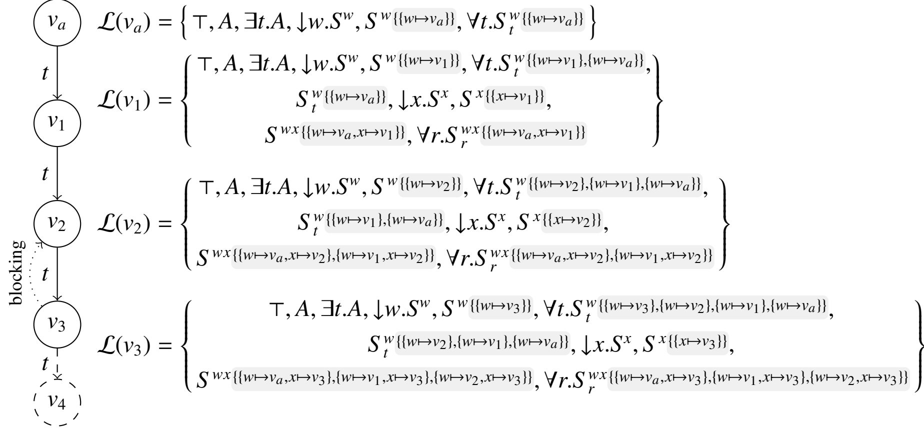

Fig. 7. Expansion blocked completion graph for Example 7 with variable mappings

Example 7. For testing entailment of Q_3 (cf. Figure 3) 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 4 with $S^w_t \sqsubseteq \forall t.S^w_t$ (cf. [11]). For the resulting axioms, the tableau algorithm creates a completion graph as depicted in Figure 7, 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, 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 $\mathsf{states}(v_3, \{w \mapsto v_3\}) = \{S^w, \forall t.S^w_t, S^x\}$ 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 3). A witness for $\{w \mapsto v_3\}$ is $\{w \mapsto v_2\}$ since $\mathsf{states}(v_2, \{w \mapsto v_2\}) = \{S^w, \forall t.S^w_t, S^x\}$ and $\mathsf{states}(v_1, \{w \mapsto v_2\}) = \{S^x\}$. Note that a variable mapping (e.g., $\{w \mapsto v_a\}$) might be the witness mapping for several variable mappings (e.g., for $\{w \mapsto v_1\}$ as well as for $\{w \mapsto v_a\}$) and vice versa (cf. third, fifth, and last row of Table 3).

It is clear that analogous propagation blocking ensures termination of a tableau algorithm with the presented extensions (as long as no new nominals are generated) since the sets of (query state) concepts associated with compatible variable mappings as used by analogous propagation blocking are bounded. In fact, the concepts that can occur in node labels are limited by the knowledge base as well as the query absorption and, hence, we eventually have variable mappings that are associated with the same set of concepts for a similar group of nodes. In addition, analogous propagation blocking also ensures that the completion graph is expanded so far that query entailment can safely be decided (again, as long as no new nominals are generated).

Table 3. Witness mappings for testing analogous propagation blocking for Example 7

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

Example 8. Let us assume that we have the following axioms in knowledge base $\mathcal{K}_3$:

$$\begin{array}{ccc} \exists s.A(a) & A \sqsubseteq \exists s^-.B & B \sqsubseteq \exists s^-.C \\ B \sqsubseteq \exists s^-.r^-.t^-.\top & C \sqsubseteq \exists t^-.A & C \sqsubseteq \exists r'.\{o\} \\ r' \circ r'^- \sqsubseteq r & & \end{array}$$

If we absorbed Q_3 (cf. Figure 3) as in Example 4 such that we have the axioms depicted in Figure 4, then we further have to "unfold" $S^{wx} \sqsubseteq \forall r.S^{wx}_r$ for the complex role inclusion $r' \circ r'^- \sqsubseteq r$, e.g., with the following axioms:

$$S^{wx} \sqsubseteq \forall r'.S^{wx}_{r'} \qquad S^{wx}_{r'} \sqsubseteq \forall r'^-.S^{wx}_r$$

By testing the consistency with the tableau algorithm considering all these axioms, we get at some point the completion graph depicted in Figure 8. Note that the shown completion graph is not fully expanded. In fact, the tableau algorithm now has to determine whether node v_5 is blocked, and if this is not the case, then the $\exists$-rule for $\exists s^-.C \in \mathcal{L}(v_5)$ must be applied, i.e., node v_6 must be constructed as shown in dashed form in Figure 8. It is clear that query Q_3 is entailed by the knowledge base. More precisely, the path $\langle v_4, v_3\rangle, \langle v_3, v_o\rangle, \langle v_o, v_6\rangle, \langle v_6, v_5\rangle, \langle v_5, v_4\rangle$ satisfies the query since the corresponding edges constitute instances of t, r', r'^-, s, s and, due to the complex role inclusion axiom $r' \circ r'^- \sqsubseteq r$, we have t, r, s, s, which are the role terms of Q_3. Note that we also have the paths $\langle w_3, w_2\rangle, \langle w_2, w_1\rangle, \langle w_1, v_2\rangle, \langle v_2, v_1\rangle$ and $\langle w_6, w_5\rangle, \langle w_5, w_4\rangle, \langle w_4, v_5\rangle, \langle v_5, v_4\rangle$, which satisfy most of the query, but do not close the required cycle. They are added to the example in order to make the blocking checking not completely trivial. In particular, if these paths were not present, then already the pairwise blocking condition could detect that v_5 is not blocked by v_2 since we would not have the concepts $S^{wxy}_s, ..., \forall s.S^{wxyz}_s$ in the label of v_5 but in the label of v_2.

For checking analogous propagation blocking, we first use the pairwise blocking conditions to find an ancestor of v_5 that is labelled with the same concepts, which is only the node v_2 in the example. As one can see, also the predecessors are labelled with the same concepts (i.e., $\mathcal{L}(v_4) = \mathcal{L}(v_1)$ and the labels of the edges from the predecessors are the same (i.e., $\mathcal{L}(\langle v_4, v_5\rangle) = \mathcal{L}(\langle v_1, v_2\rangle)$). Now we consider the to be tested node, its predecessor and the nominal node and check for each variable mapping whether there

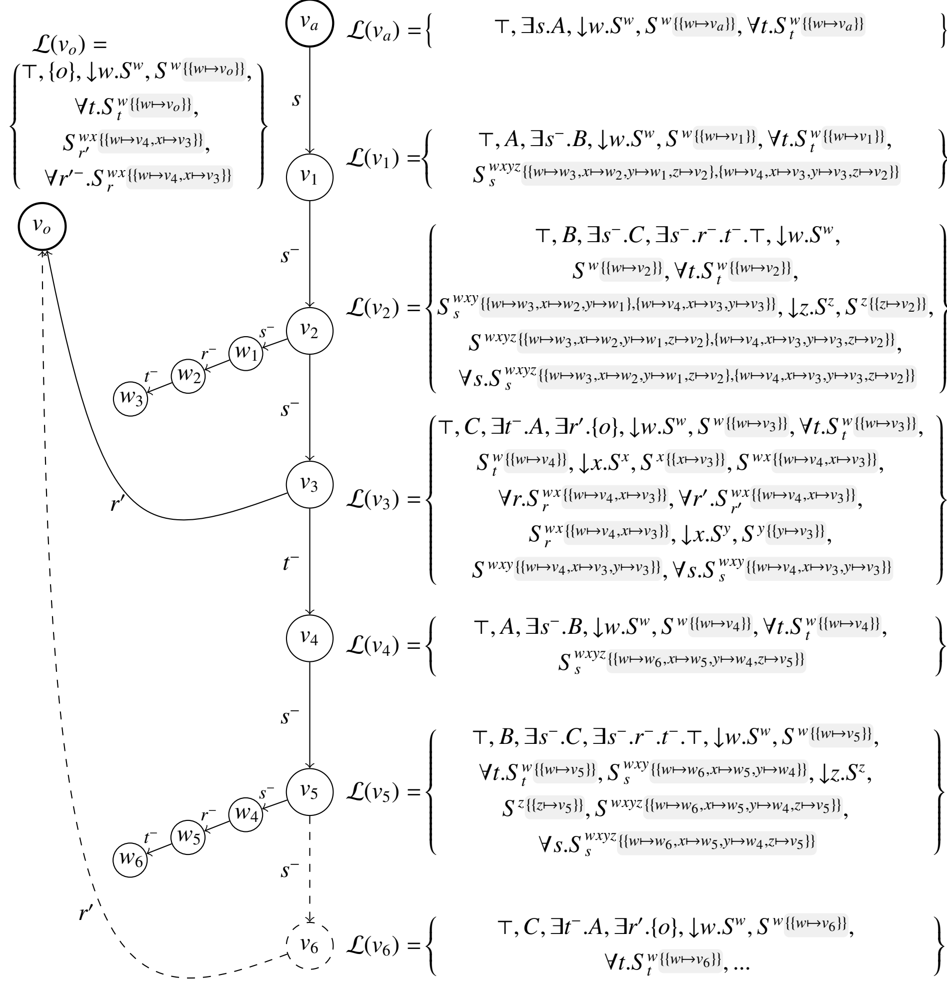

Fig. 8. Completion graph for Example 8 for testing blocking in presence of nominals and propagated variable mappings

exists a witness mapping for the other nodes, i.e., the blocking candidate, its predecessor and the nominal node (and vice versa). For example, if we consider the variable mapping $\{w \mapsto v_5\}$ for node v_5 (with predecessor v_4 and nominal node v_o), then we can identify $\{w \mapsto v_2\}$ as witness mapping since we have the following states:

$$\mathsf{states}(v_5, \{w \mapsto v_5\}) = \mathsf{states}(v_2, \{w \mapsto v_2\}) = \{S^w, \forall t.S_t^w, S^z\},$$
$$\mathsf{states}(v_4, \{w \mapsto v_5\}) = \mathsf{states}(v_1, \{w \mapsto v_2\}) = \emptyset,$$
$$\mathsf{states}(v_o, \{w \mapsto v_5\}) = \mathsf{states}(v_o, \{w \mapsto v_2\}) = \emptyset.$$

However, there is no witness mapping for $\{w \mapsto v_4\}$ occurring in node v_4 with

$$\begin{aligned}
\mathsf{states}(v_5, \{w \mapsto v_4\}) &= \{S^z\},\\
\mathsf{states}(v_4, \{w \mapsto v_4\}) &= \{S^w, \forall t.S^w_t\},\\
\mathsf{states}(v_o, \{w \mapsto v_4\}) &= \{S^{wx}_{r'}, \forall r'^-.S^{wx}_r\}.
\end{aligned}$$

In addition, the mapping $\{w \mapsto v_4, x \mapsto v_3, y \mapsto v_3\}$ occurring in node v_2 with

$$\begin{aligned}
\mathsf{states}(v_2, \{w \mapsto v_4, x \mapsto v_3, y \mapsto v_3\}) &= \{S^{wxy}_s, S^z, S^{wxyz}, \forall s.S^{wxyz}_s\},\\
\mathsf{states}(v_1, \{w \mapsto v_4, x \mapsto v_3, y \mapsto v_3\}) &= \{S^{wxyz}_s\},\\
\mathsf{states}(v_o, \{w \mapsto v_4, x \mapsto v_3, y \mapsto v_3\}) &= \{S^{wx}_{r'}, \forall r'^-.S^{wx}_r\}
\end{aligned}$$

has no witness mapping w.r.t. the tested node, its predecessor and the nominal node. Consequently, the expansion of node v_5 cannot be blocked w.r.t. analogous propagation blocking since it is correctly detected that variable mappings from the tested node and the blocker candidate are propagated differently over the nominal node.

One may wonder, whether the analogous propagation blocking condition is less restrictive than the comparison of concepts of foldings. In particular, we no longer check whether some variables are bound/mapped to the same node (as long as this node is not the considered node, nor its predecessor, nor a nominal node), which basically means that we forget to some extent how the query is satisfied, i.e., which particular folding has been found. We also have this effect if we omit the creation of certain bindings for the propagation and/or if we remove bindings for variables that are no longer relevant after join operations. However, one can show that this can also be realised by rewriting the foldings with fresh atomic concepts and corresponding concept definitions in the knowledge base. For example, possible foldings of the query $Q_3 = \{t(w, x), r(x, y), s(y, z), s(z, w)\}$ (cf. Figure 3) are $\exists(t \sqcap s^-).\exists(r \sqcap s^-).\top$ and $\exists(t \sqcap r^-).\top \sqcap \exists(s \sqcap s^-).\top$. Note that here both concepts express the foldings from the position of the variable w. If we consider an extension of the query with a second cycle, e.g., by adding the terms $f(w, w'), t'(w', x'), r'(x', y'), s'(y', z'), s'(z', w')$, then we can build combinations of these foldings, i.e., we have overall four concepts for representing foldings. If we, however, add $F \equiv (\exists(t \sqcap s^-).\exists(r \sqcap s^-).\top) \sqcup (\exists(t \sqcap r^-).\top \sqcap \exists(s \sqcap s^-).\top)$ to the knowledge base, we can express the concepts for foldings of the second cycle independently, which, for example, results in $\exists f^-.F \sqcap \exists(t' \sqcap s'^-).\exists(r' \sqcap s'^-).\top$ and $\exists f^-.F \sqcap \exists(t' \sqcap r'^-).\top \sqcap \exists(s' \sqcap s'^-).\top$. Let us now assume that we detected for a (blockable) node that the sub-concept $\exists f^-.F$ is satisfied for it, so we add it to the node label. If we now consider blocking for this node by comparing concepts for foldings, then only the concept $\exists f^-.F$ is considered for checking with the pairwise blocking condition whether an ancestor node has the same label and, hence, we ignore how the query is satisfied for the blocking test. Omitting (or eliminating) bindings in variable mappings can be seen analogous to the replacement of sub-concepts of foldings with (fresh) atomic concepts from the knowledge base in the sense that we lose the ability to (locally) restore how the query is satisfied. Since the corresponding parts of the query are, however, already completed, this is not problematic for blocking.

Of course, considering all nominal nodes for each blocking test could be quite problematic in practice. However, we can easily find optimisations that significantly restrict

the number of such tests and the number of nominal nodes that have to be considered in these tests. In particular, we only have to consider analogous propagation blocking with the sets of concepts associated with compatible variable mappings if all other (pairwise) blocking conditions hold. We can further avoid considering the nominal nodes in some cases by using a pre-test that compares the mappings only between the tested node v and the potential blocker node w, i.e., we first check whether each variable mapping μ for v has a witness mapping μ' for w (and vice versa) such that at least $\mathsf{states}(v, \mu) = \mathsf{states}(w, \mu')$ holds. If there are witness mappings for all variable mappings, then the test can be extended to also consider/include the predecessors for the states conditions and, then, the relevant nominal nodes. In addition, we can track the usage of nominal nodes, i.e., we save for each node those nominal nodes with which the node or a descendant is connected. For the blocking test, we can then restrict the consideration to those nominal nodes that have been collected for the potential blocker node with such a tracking technique. Moreover, we can index for which nominal nodes we have (which) variable mappings associated with concepts. Then, for a variable mapping considered in a blocking test, we can easily resolve (potentially) relevant nominal nodes for compatible mappings.

4.3 Absorption Optimisations

The query absorption algorithm presented in Section 4.1 leaves room for improvements. In fact, the creation and the propagation of variable mappings causes additional overhead in comparison to the processing of ordinary concepts and axioms and, thus, we should try to reduce it as much as reasonably possible. Although there exist approaches to optimise the handling of variable mappings directly in the tableau algorithm (e.g., by using a representative propagation of mappings [27], batch-based processing of propagations [27], etc.), there are also several starting points directly in the absorption algorithm as discussed in the following.

First of all, we should try to avoid the creation of bindings for variables that are not really used, i.e., variables that are not required for joining different propagations. This is especially the case for variables that occur in only two role terms. For example, query Q_3 from Figure 3 consists of the terms $t(w, x)$, $r(x, y)$, $s(y, z)$, $s(z, w)$ and has the four variables w, x, y, z, but the join in the final binary inclusion axiom $S^{wxyz} \sqcap S^{w} \sqsubseteq S^{wxyzw}$ only relies on the variable w, i.e., bindings for x, y, and z are not required. If we omit the creation of binder concepts for these variables, then we can get an absorption of the query consisting of only the following axioms:

$$\top \sqsubseteq {\downarrow}w.S^{w} \qquad S^{w} \sqsubseteq \forall t.S^{w}_{t} \qquad S^{w}_{t} \sqsubseteq \forall r.S^{w}_{tr}$$
$$S^{w}_{tr} \sqsubseteq \forall s.S^{w}_{trs} \qquad S^{w}_{trs} \sqsubseteq \forall s.S^{w}_{trss} \qquad S^{w}_{trss} \sqcap S^{w} \sqsubseteq S^{ww}_{trss}$$
$$S^{ww}_{trss} \sqsubseteq \bot$$

Since the query state concepts are technically only required for joins, we can even omit most of them. As a result, we can absorb the query with just the following three axioms:

$$\top \sqsubseteq {\downarrow}w.S^{w} \qquad S^{w} \sqsubseteq \forall t.\forall r.\forall s.\forall s.S^{w}_{trss} \qquad S^{w}_{trss} \sqcap S^{w} \sqsubseteq \bot.$$

Although some optimisations allow for handling several variable mappings at the same time (e.g., with the representative propagation of variable mappings, we propagates

one representative "element" instead of all mappings associated with a concept fact separately [27]), it can, however, not always be avoided that repeatedly new variable mappings are propagated to the same concept facts such that the rules must also repeatedly be reapplied. Consequently, each additional (query state) concept can cause some significant overhead and, thus, should be avoided, too. Note that the analogous propagation blocking condition, as defined in Section 4.2, compares all concepts associated with compatible variable mappings, i.e., we do not rely on the query state concepts for blocking since also universal restriction with associated mappings in the label of nominal nodes are correctly considered. After absorbing the query, we can therefore go through the axioms and eliminate all query state concepts that are not required, i.e., if they are neither used to imply multiple other concepts nor in binary inclusion axioms for joins. In contrast, a simple version of the optimisation that omits/avoids binder concepts can directly be integrated in the absorption algorithm by modifying the absorbRT function (cf. Algorithm 3) such that it checks whether the neighbouring variable (to which is propagated) occurs in only one other role term (and only once). If this is the case, then we can directly append the propagation over that role term and we can repeat this step as long as we have such variables. Concept terms for these "omitted" variables can still be handled by appropriately creating and storing query state concepts and, then, calling absorbCT from the absorbRT function.

Another reduction of propagation work in completion graphs can be achieved by "delaying" the addition of the first binder concept instead of just implying it from $\top$ such that it is added to all node labels. In particular, we can (partially) pre-absorb (parts of) a "spanning tree" from the cyclic query to trigger the first binder concept. If we consider query Q_3 again, then such a (partial) pre-absorption could yield the axiom $\top \sqsubseteq \forall t^-.\forall r^-.\forall s^-.\forall s^-.F$, where F is a fresh atomic concept that can subsequently be used to imply the first binder concept (e.g., with $F \sqsubseteq \downarrow w.S^w$). With this axiom, F and, thus, the binder concept is automatically added to only those node labels where the query could actually be satisfied, i.e., where all required concepts and roles are present in the neighbourhood. Note that some naive absorption algorithms may force non-deterministic decisions for some complex concept expressions (e.g., $\geqslant 3\, r.B$), which is usually not desired for just identifying candidates. In fact, we usually try to avoid and/or delay non-deterministic decisions until they are indeed required, which can, for example, be achieved with "partial absorption" techniques [26] (e.g., by absorbing $\exists r.B$ instead of $\geqslant 3\, r.B$). Also note that the partial absorption techniques can be used to improve the actual propagation for complex concept terms. If we consider the term $\geqslant 3\, r.B(x)$ again, then the query absorption algorithm simply creates a decision of the form $S^{x_1 \ldots x_n x} \sqsubseteq\ \leqslant 2\, r.B \sqcup F^x_{\leqslant 2 r.B}$ (cf. Algorithm 2), which can further be "improved" with partial absorption techniques by replacing it with the axioms $B \sqsubseteq \forall r^-.F'$ and $S^{x_1 \ldots x_n x} \sqcap F' \sqsubseteq\ \leqslant 2\, r.B \sqcup F^x_{\leqslant 2 r.B}$, where the processing of the disjunction is now further delayed/avoided until the node has an r-neighbour with B in its label. In some cases, the disjunction can even completely be avoided by further rewriting/absorbing the obtained axioms. For example, the query absorption algorithm generates for a concept term $\exists r.B(x)$ an axiom of the form $S^{x_1 \ldots x_n x} \sqsubseteq \forall r.\neg B \sqcup F^x_{\exists r.B}$, but we can easily further absorb this axiom by "moving" $\forall r.\neg B$ to the left-hand side. In fact, by adding $B \sqsubseteq \forall r^-.F'$

to the knowledge base, we can replace the previous axiom with $S^{x_1 \ldots x_n x} \sqcap F' \sqsubseteq F^x_{\exists r.B}$, which can deterministically be processed by the tableau algorithm.

As already mentioned, any useless or redundant propagation (of variable mappings) creates some overhead and, thus, should be avoided. The presented absorption algorithm, however, separately propagates over each role term step by step and, if the next query term is not at the position of the current variable, then we may even build redundant propagations over already absorbed role terms. This can also be improved with a slightly more sophisticated absorption algorithm. In fact, if we directly absorb all those role terms that contain the current variable and do not "disconnect" the remaining variables if the absorbed role terms are removed, then we can get an absorption that propagates over each role term only once. More precisely, we say that two variables x and z are *connected* w.r.t. the set of query terms Q if there exist role terms such that $r_1(x, y_1), r_2(y_1, y_2), ..., r_{n-1}(y_{n-1}, y_n), r_n(y_n, z) \in Q$. Now, let Q be the set of unprocessed query terms. Then, we select one variable z (preferably one for which already a binder concept exists) and a role term $q \in Q$ of the form $r(z, x)$ such that x is still connected w.r.t. $Q \setminus \{q\}$ to every variable y with $s(z, y) \in Q \setminus \{q\}$. We now absorb the role term q and remove q from Q. If there are more such non-disconnecting role terms for z, we absorb them analogously and then we continue the absorption by repeating these steps. Note that this kind of absorption does not relay on a current variable, but may introduce several propagation starting points or branches for the query, but this is neither problematic for the tableau rules nor for the blocking technique. In fact, the blocking condition is based on the compatibility of variable mappings, which also covers cases, where different propagations start independently and are combined/joined at some point. For example, if the variable mapping $\{x \mapsto v_a\}$ is associated with a concept on the nominal node v_a and $\{y \mapsto v_b\}$ with a concept on the nominal node v_b and we consider blocking for a node that has a concept associated with the mapping $\{x \mapsto v_a, y \mapsto v_b, z \mapsto v_1\}$, then the analogous blocking technique correctly considers that the propagations for $\{x \mapsto v_a, y \mapsto v_b, z \mapsto v_1\}$ could go over these nominal nodes since $\{x \mapsto v_a\}$ as well as $\{y \mapsto v_b\}$ are compatible with $\{x \mapsto v_a, y \mapsto v_b, z \mapsto v_1\}$.

As one can observe, the absorption handles a role term by propagating existing bindings over the corresponding role to a new "state concept" and then by separately joining it with the last state for the other variable. This is suboptimal since the bindings could also be propagated to many nodes for which we do not have compatible variable mappings, i.e., it may not be possible to further continue the propagation on many of these nodes. Theoretically, one can avoid this by utilising a combined propagation and joining rule, but this is quite atypical for (classical) tableau algorithms. In fact, the applicability of expansion rules can usually simply be checked for those nodes for which a concept is added and/or an edge is modified. A combined propagation and joining rule, which would basically only propagate to nodes that have compatible variable mappings associated with a certain concept in their label, clearly does not follow this idea and, hence, can be difficult to implement. In addition, the handling of complex roles seems to be much simpler with this separated propagation step since we can unfold/expand the universal restrictions as required w.r.t. the automata for role inclusion axioms. Also note that the propagation overhead of variable mappings to potentially many irrelevant nodes

can be addressed (to some extent) by propagating variable mappings in a representative way [27].

5 Optimized Query Answering Reduction

Instead of naively grounding conjunctive queries with answer variables to (exponentially) many query entailment checks (e.g., by adding nominal concept terms), we can simply modify the presented absorption algorithm such that it delivers (candidates for) answers. For this, we assume that the knowledge base is extended by concept assertions of the form $O(a)$ for each individual a, where O is a fresh atomic concept that allows for distinguishing known and anonymous individual in the knowledge base. Then, we extend the query with concept terms of the form $O(x)$ for each answer variable x and absorb the query as presented in Section 4. However, we do not imply $\bot$ from the last query state concept that is generated by the algorithm. In addition, we have to ensure that binder concepts for the answer variables are always created, i.e., we appropriately adapt the optimisations of Section 4.3. Now, 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, i.e., we extract for each variable mapping to which individual nodes the answer variables are bound. By replacing the nodes with the corresponding individuals, we get the answer candidates in form of tuples. 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 certainly an answer. For the non-deterministically derived/propagated variable mappings, we have to verify the obtained answer candidates with corresponding query entailment checks. Note that there exist several techniques that cache the completion graph of the consistency test [23,28], i.e., it is usually not necessary to rebuild the entire completion graph for each query propagation/entailment check. Nevertheless, each test causes some overhead and, with the extraction of propagated variable mappings, the number of required entailment checks can significantly be reduced. In particular, 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.

5.1 Instance Query Absorption

Due to the one-pass behaviour, the reduction to such propagation tests is often also beneficial for non-cyclic queries, for which, theoretically, a simple rolling-up of the query is possible. In fact, a rolling-up yields complex concept terms, but in order to compute the answers of a complex concept term with an answer variable, a tableau-based reasoner typically has to test, for each individual, whether it is an instance of that concept. Instead of using such a naive reduction to many instance tests, we can also build these propagation tests similar to (cyclic) queries with several answer variables. In fact, we can simply absorb the tree-shaped structure of the query and imply a fresh atomic

marker concept F. Now, by building or extending a completion graph for the extension of the knowledge base that additionally contains the axioms from the absorption, we propagate F to the labels of those nodes that potentially satisfy the query, i.e., by identifying the corresponding individuals of these nodes and whether F has been derived deterministically for them, we get lower and upper bounds for the corresponding instance retrieval task. Even if the some complex concept terms in the query are not "completely absorbable", the obtained upper bound from such a propagation test can significantly help in reducing the number of required instance tests.

For example, a query with the answer variable x and the query terms $A(x), r(x,z)$, $s(z,z_a), \{a\}(z_a)$, $s(z,z_b), \{b\}(z_b)$ can be rolled-up such that only the concept term $(A \sqcap \exists r.(\exists s.\{a\} \sqcap \exists s.\{b\}))(x)$ remains, i.e., we have an instance retrieval task w.r.t. the concept $A \sqcap \exists r.(\exists s.\{a\} \sqcap \exists s.\{b\})$. By absorbing the concept/query, we get the following axioms:

$$\{a\} \sqsubseteq \forall s^{-}.F_1 \qquad \{b\} \sqsubseteq \forall s^{-}.F_2 \qquad F_1 \sqcap F_2 \sqsubseteq \forall r^{-}.F_5 \qquad F_5 \sqcap A \sqsubseteq F$$

If we construct a fully expanded and clash-free completion graph for the knowledge base with these axioms, then F in the label of a node for an individual a indicates that a could be an instance of the absorbed concept. Moreover, if F has been derived deterministically for that node, then we know that a is certainly an instance of the concept and, if F has not been derived for a node that represent an individual a, then a cannot be an instance of the absorbed concept. If we have a complex concept that cannot (easily) be absorbed completely, e.g., $\forall r.B \sqcap \exists s.A$, then we can partially absorb it (i.e., we absorb the part of the concept that is absorbable, e.g., $\exists s.A$) and we also consider the individuals for which the concept F has been derived deterministically as possible instances, i.e., we have to verify all determined individuals with instance tests.

5.2 Integration of Realisation Results

Query terms with only answer variables (and only atomic concepts) clearly correspond to (atomic) concept and role instance retrieval queries. Since these (atomic) instances are also computed by the (concept/role) realisation task, we can use the realisation results to resolve these parts of the query. In fact, realisation is a well-known reasoning task that is already implemented in many state-of-the-art systems and several optimisations have been developed that make realisation quite efficient. Hence, an interesting opportunity for improving query answering can be the integration of realisation results. In particular, the concept and role instances computed by the realisation process are typically cached and, therefore, they can easily be reused for subsequent queries.

The presented query answering approach can easily be extended to utilise realisation results. First, we identify those parts of a query that can be resolved via realisation results, i.e., query terms with only answer variables and atomic concepts (if they are concept terms). Next, we resolve the answers for these query terms using the computed instances by the realisation process, which are then interpreted as an upper bound for the entire query. Note that separately resolving each query terms can lead to different answers for a variable, for which then an intersection over all answers for a variable gives the upper bound. 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. With this technique, we reduce the propagation work in the completion graph in different ways. On the one hand, we do not have to propagate over role terms that are already resolved via realisation results. On the other hand, we do only create certain bindings, which also restricts how and to which nodes variable mappings are propagated.

The realisation process is often implemented as an on demand service, i.e., only those instances are computed that are actually required and/or requested. In other words, it lazily computes the concept and role instances. This is particularly important for role realisation since determining all instantiations of all complex roles can require a lot of instance tests and even storing the (possible) instances (naively) can require a lot of memory. In fact, a transitive role t is typically realised by testing for each individual a to which other individuals a marker concept F can be propagated, e.g., by asserting $\forall t.F$ to a, where we additionally have the axiom $F \sqsubseteq \forall t.F$ in the knowledge base. Again, if F has been propagated/derived non-deterministically, then we have to verify the extracted role instances by additionally asserting $\neg F$ to the related individual. If all or many individual are (indirectly) connected via such a transitive role, then we not only have many tests, but also that each test is quite expensive (since we always propagate to all related individuals).

It is often desired to extend the lazy processing approach of the realisation service to the query answering service such that long preparation times can be avoided. One state-of-the-art technique to minimize the required instance computations from the realisation process is based on pruning upper bounds for sub-queries [7]. As we show in the following example, our query answering approach can nicely be integrated into this optimisation. Roughly speaking, we also prune the upper bounds for answer variables with the variable mappings from a propagation test.

Example 9. Let us assume that we have a query $Q_5(\langle x_a, x_b, x_c\rangle) = \{r_1(x_a, x_b), r_2(x_b, x_c), s_1(x_a, z), s_2(z, x_c)\}$. Since the first two query terms have only answer variables, they can be resolved via realisation results, but this would require that the roles r_1 and r_2 are realised by the reasoner. Instead of separately computing all instances of r_1 and r_2, one can extract the instance candidates for these roles, e.g., from the (cached) completion graph of the consistency test. Let us assume that $\{\langle a_1, b_1\rangle, \langle a_1, b_2\rangle, \langle a_2, b_3\rangle\}$ is the set of possible instances for r_1 and $\{\langle b_1, c_1\rangle, \langle b_2, c_2\rangle, \langle b_3, c_2\rangle, \langle b_4, c_1\rangle\}$ the set of possible instances for r_2 that can be found in a completion graph. Clearly, $\{a_1, a_2\}$ can be seen as an upper bound for x_a and $\{b_1, b_2, b_3, b_4\}$ an upper bound for x_b, whereas $\{c_1, c_2, c_3\}$ represents the upper bound for x_c. By considering the upper bounds for answer variables of the sub-queries, which is the set $\{a_1, a_2\}$ w.r.t. x_a and $\{b_1, b_2, b_3\}$ w.r.t. x_b for the sub-query $r_1(x_a, x_b)$ and the set $\{b_1, b_2, b_3, b_4\}$ w.r.t. x_b and $\{c_1, c_2, c_3\}$ w.r.t. x_c for the sub-query $r_2(x_b, x_c)$, we can further identify individuals that cannot be (part of) answers for the entire query (e.g., b_4) and, hence, we can "prune the bounds" for the sub-query. In particular, we do not need to know whether $\langle b_4, c_1\rangle$ is an instance of r_2, i.e., the realisation process does not have to compute it. By absorbing the remaining query, e.g., $\{s_1(x_a, z), s2(z, x_c)\}$ with an axiom of the form $\downarrow x_a.\forall s.\forall s.S_{ss}^{x_a}$, and by building a corresponding completion graph, where we restrict the $\downarrow$-rule to bind x_a only to nodes that represent the individual a_1 or a_2, we can identify to which nodes the variable map-

pings have been propagated and we can use this information to further prune the upper bounds. For example, if we find that the variable mappings $\{x_a \mapsto a_1\}$ and $\{x_a \mapsto a_2\}$ have only been propagated to the node c_1, then we further do not require from the realisation that $\langle b_2, c_2 \rangle$ and $\langle b_3, c_2 \rangle$ are included in the instance computations of r_2 . In turn, it is even possible to prune b_2 and b_3 and, thus, $\langle a_1, b_2 \rangle$ and $\langle a_2, b_3 \rangle$ can be ignored for the instance computation of r_1. Now, we can build a query evaluation plan and request realization results based on the determined mappings for answer variables so far and the pruned upper bounds. If we start, for example, at r_1, then we request realisation of the remaining possible instances for r_1, i.e., $\langle a_1, b_1 \rangle$, and we integrate the results in the answer mappings. Let us assume that $\langle a_1, b_1 \rangle$ is indeed an instance of r_1, then we can continue with the next query term, e.g., $r_2(x_b, x_c)$, for which we then request realisation of the remaining possible instance for r_2 (i.e., $\langle b_1, c_1 \rangle$) based on the upper bounds for the variables and the collected mappings. Again, let us assume that $\langle b_1, c_1 \rangle$ is indeed an instance of r_2, then we have the mapping $\{x_a \mapsto a_1, x_b \mapsto b_1, x_c \mapsto c_1\}$ as an answer candidate for the query. Last but not least, we may have to verify with an entailment test whether the (sub-)query $\{s_1(a_1, z), s2(z, c_1)\}$ actually holds (if the corresponding variable mapping in the propagation test has been derived non-deterministically).

As one can see, the separated propagation of variable mappings fits nicely into existing pruning approaches such that entailment tests (of role/concept instantiations and other queries with existential variables) can often significantly be reduced. Although it would also be possible to completely absorb and handle the query with the presented query answering approach, the integration and utilisation of the realisation process has the benefit that the results for the corresponding parts of the query can be cached more easily and, as mentioned, there exist special optimisations for the realisation process (e.g., summarization and abstraction techniques) such that a separate evaluation can even be faster.

6 Implementation and Experiments

We implemented the presented query answering approach into the tableau-based reasoning system Konclude [29], which supports the DL $\mathcal{SROIQ}$ with nominal schemas, i.e., an extension of the nominal constructor by variables that allows for representing rule-based knowledge directly in ontologies. Axioms with these nominal schemas are also absorbed in Konclude such that simple axioms are obtained that can efficiently be handled by the tableau algorithm. In fact, this absorption-based query answering approach can be seen as an extension of the nominal schema absorption technique, where 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 5. Nevertheless, both techniques use to a significant extent the same implementations for propagating variable mappings through completion graphs.

Konclude also implements a completion-based saturation procedure that is only complete for less expressive Horn-DLs (e.g., Horn-$\mathcal{SHIQ}$) and is primarily used to

assist the tableau algorithm. In fact, creating and propagating variable mappings is only implemented for the tableau algorithm. In order to avoid reconstructing the entire completion graph for each instance test (e.g., for realisation), Konclude uses a sophisticated completion graph caching technique [28], which is also used for improving the query entailment tests. In particular, Konclude reuses the last deterministic completion graph from the initial consistency check for all subsequent satisfiability/consistency tests and checks with certain criteria whether and which nodes with non-deterministically derived consequences have to be expanded again, e.g., due to different non-deterministic decisions in some extended/processed part of the completion graph. In other words, Konclude reprocesses only the non-deterministic expansions of those (nodes for) individuals for which an expansion as in the initial completion graph from the consistency test cannot easily be guaranteed. If there are too many individuals such that caching the entire completion graph does not make sense/requires too much memory, then Konclude falls back to a representative variant of the completion graph caching technique. More precisely, Konclude stores/caches the derived data for individuals from the consistency check in a representative way and reuses the cached data to check to which individuals the processing of the current completion graph has to be extended. This "representative (completion graph) caching" is less precise than the ordinary completion graph caching, i.e., more individuals may have to be reprocessed in subsequent tests, but it clearly requires less memory, i.e., it is a trade-off between memory consumption and reasoning time. Since ontologies with large ABoxes often have many individuals with similar structures, we can usually neglect the required memory for storing the derived data in the "representative cache". Hence, we configured Konclude to always fill the representative cache with derived data from the consistency check (even if the ordinary completion graph caching technique is used) and to use the cache as an index for quickly resolving instance candidates for instance checking/retrieval tasks. This directly enables a simple form of the sub-query bound pruning such that less candidates have be considered for the query. In fact, we extract for each answer variable local restrictions and use them to resolve candidates for that variable via the representative cache. To be more precise, we extract for an answer variable concepts of the form $A_1 \sqcap ... \sqcap A_n \sqcap \exists r_1.E_1 \sqcap ... \sqcap \exists r_m.E_m$ with $E_i = \top$ or $E_i = \{a\}$ or $E_i = O$ (for $1 \leq i \leq m$), where O is the special atomic concept that is asserted to all individuals in the knowledge base. If the cached labels in the representative cache cover several expressions of the extracted concept, then we can simply look for labels that contain all required (sub-)concepts expressions and, hence, by only using the individual associated with these labels, we already get a limited set of candidates for the variable. For an example, let us consider a query with the terms $r_1(x, y)$, $s(x, y_a)$, $\{a\}(y_a)$, and $r_2(x, z)$, where z is the only existential variable. For the variable x, we can "extract" the concept $\exists r_1.O \sqcap \exists s.\{a\} \sqcap r_2.\top$, which basically captures the local restrictions of x. Now let us assume that we stored/cached which outgoing and incoming roles have been derived for an individual (in the consistency check) and whether it is related to a known or an anonymous individual w.r.t. that role. From the extracted concept expression, we know that the answers of the query can consist only of those tuples, for which x corresponds to individuals that are in relation w.r.t. the roles r_1, r_2, and s. With the cache, we can identify all "representative labels" that contain these roles and we can use the associated

individuals as an upper bound for x. Since we also stored for which roles individuals are related to other known individuals or only to anonymous individuals, we can further prune the upper bound appropriately. However, since not all information is stored in the same "representative labels", we may have to do sort-merge/hash joins over the associated individuals of these cached labels. Clearly, managing all derived data together could easily prevent a representative storage since less individuals may share the same representative label, i.e., the "summarisation" can be less effective. Therefore, we manage different data separately, i.e., we have one representative label for the derived concepts (and whether they have been derived deterministically), one for derived roles, one for (potentially) same individuals, etc. By using these different types of representative labels, we can then resolve the individuals that certainly or potentially instantiate the extracted concepts.

Unfortunately, Konclude does not fully support incremental reasoning. In particular, we cannot simply add new (unfolding) axioms (as introduced by our query absorption algorithm) to the knowledge base and continue the processing of the completion graph. For this, it would be required to store for each atomic concept A those nodes that have A in their label such that the addition of an unfolding axiom with A on the left-hand side can efficiently be handled by identifying the nodes with A in their label and checking whether an unfolding rule is applicable for them. In contrast, it is also quite expensive to simply check for all nodes whether a new unfolding axiom has to be considered. Therefore, we rewrite the axioms from the absorption such that corresponding marker concepts are first propagated to the relevant nodes, e.g., from root/individual nodes.

For example, a query consisting of the only term $\exists r.(A \sqcap B)(x)$ with the answer variable x can be absorbed with the axioms $A \sqcap B \sqsubseteq \forall r.F'$ and $F' \sqcap O \sqcap F$. By building/extending a completion graph such that the new axioms are considered, we directly propagate the fresh marker concept F to the nodes of individuals that constitute a (possibly) answer of the query. However, as mentioned, it would be necessary to detect/identify (blockable) nodes that are labelled with A or B such that the rule application for $A \sqcap B \sqsubseteq \forall r.F'$ can be checked. In contrast, we rewrite the axioms from the absorption as follows:

$$F_0 \sqcap O \sqsubseteq \forall r.F_1 \qquad F_1 \sqcap A \sqsubseteq F_2 \qquad F_2 \sqcap B \sqsubseteq \forall r^-.F_3 \qquad F_3 \sqcap F_0 \sqsubseteq F_4$$

With these axioms, we can simply add F_0 to "relevant" individual nodes and extract to which individuals the concept F_4 has been propagated for the expanded completion graph to get answer candidates for the query. In fact, we have for each unfolding axiom at least one fresh atomic (marker) concept that has been propagated to the nodes, i.e., it is clear for which nodes the rule application has to be checked. The addition of F_0 to all individual/root nodes is obviously also not efficient, but we can often identify a much smaller set of relevant individuals by analysing the concept expression. In particular, we know from $\exists r.(A \sqcap B)$ that potentially relevant individuals must have an r-successor, which can be resolved, e.g., via the representative cache in Konclude. Note that the last axiom from the rewriting above is not necessary for this example, but a join with the "last marker concept of a level" becomes necessary if there are several existential restrictions, e.g., the absorption and subsequent rewriting of $\exists r.A \sqcap \exists s.B$ results in the

following axioms:

$$F_0 \sqcap O \sqsubseteq \forall r.F_1 \quad F_1 \sqcap A \sqsubseteq \forall r^-.F_2 \quad F_2 \sqcap F_0 \sqsubseteq \forall s.F_3 \quad F_3 \sqcap B \sqsubseteq \forall s^-.F_4 \quad F_4 \sqcap F_2 \sqsubseteq F_5$$

Again, F_0 can be added to relevant individuals such that F_5 indicates answer candidates in a fully expanded completion graph. Roughly speaking, the joins with the last marker concepts of a level (i.e., $F_1 \sqcap A \sqsubseteq \forall r^-.F_2$ and $F_4 \sqcap F_2 \sqsubseteq F_5$) ensure that we check whether it has been propagated back to nodes for which the previous conditions also hold. Otherwise, marker concepts could be propagated to other individual nodes that only satisfy the concept expressions partially. If no relevant individuals can be identified or it is unclear how we have to propagate, e.g., for queries with only existential variables, then we use the universal rule in the rewriting such that indeed all nodes are checked. Clearly, the rewriting can also directly be integrated into the absorption algorithm, as realised for Konclude.

At the moment, Konclude may not terminate for $\mathcal{SROIQ}$ knowledge bases if the absorption of the query leads to propagations over new nominal nodes. In principle, it is easily detectable if much more new nominal nodes are repeatedly created (in comparison to the consistency check) and the analogous propagation blocking condition does not hold but the pairwise blocking conditions do. In this case, we could stop the completion graph construction and print a warning that the results may be incomplete. However, this does not seem to be very problematic in practice. In fact, there is only a limited number of ontologies that use all language features (i.e., nominals, inverse roles, and cardinalities/functional roles) that lead to new nominal nodes. For example, the ORE2015 dataset contains 1920 ontologies (with simple ontologies already filtered out), but only 399 (36 with $\mathcal{OIQ}$, 281 with $\mathcal{OIN}$, and 82 with $\mathcal{OIF}$) use all problematic language features. However, even if all of these language features are used, new nominal nodes are often not enforced for these ontologies. In particular, 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. Nevertheless, even if new nominals are generated, it would further be required that the query propagates differently over these new nominal and blockable nodes such that blocking cannot be established. Altogether, issues with new nominals hardly seem problematic for real-world ontologies as well as queries and it is much more likely that the system does not terminate due to other reasons.

To facilitate the evaluation of Konclude and the integrated query answering approach, we implemented a simple SPARQL HTTP server that can process some SPARQL basic graph patterns (BGPs) and has a limited support for a few SPARQL Update commands (e.g., CREATE, LOAD, DROP, INSERT, etc.). However, only those BGPs with conjunctive queries can be handled (e.g., in ASK or SELECT queries) that have only variables for individuals and only refer to atomic concepts/classes. We also integrated the Redland RDF Libaries[1] into Konclude for parsing and processing RDF datasets (with Raptor) and we plan to use the provided SPARQL engine (Rasqual) to support more SPARQL features in the future (such as UNION, OPTIONAL, etc.). The source

[1] http://librdf.org/

code, the evaluation data, results, and a Docker image (koncludeeval/abqa) for easily reproducing the evaluations, are available online [25].

6.1 Query Entailment Checking Experiments

Due to the lack of systems that are capable of fully checking entailment of conjunctive queries for more expressive DLs, there are also no suitable benchmark ontologies and queries. In order to, nevertheless, get an idea of the performance of the presented query entailment checking approach, we identified interesting ontologies and hand-crafted or generated non-trivial queries for them. More precisely, we gathered ontologies from well-known repositories that use most of the language features of $\mathcal{SROIQ}$, contain at least 100 individuals, use nominals in existentially restricted concepts, and are not completely trivial for Konclude, but also not too difficult w.r.t. standard reasoning tasks. The latter is also required for the generation of queries. In fact, we deactivated several optimisations of Konclude (e.g., anywhere blocking, caching techniques) such that the consistency check builds a completion graph where the generation of repeating structures is rather lately blocked such that cycles can easily be found. Then we randomly picked one node and tried to build a cyclic query by introducing variables for nodes and by adding role as well as concept terms for concept and role facts as they are derived in the completion graph for these nodes. Altogether, we crafted/generated 50 queries for each ontology, where each query contains at least one, but often even several cycles. Besides generating queries from real cycles (i.e., each node represents at most one variable), we also generated queries that go several times over the same nodes/edges (by using different concepts/roles in the labels for the concept/role terms), i.e., queries that may only be entailed in folded form. If the randomly picked concepts and roles from the labels have been derived non-deterministically, then the query may not be entailed. However, Konclude can handle most ontologies mainly deterministically due to the sophisticated absorption technique such that most queries are indeed entailed. Generating other queries that are certainly not entailed is non-trivial since reasoners often collect statistics of the occurrences of concepts and roles (e.g., how often they are derived deterministically, how often/whether they are derived for blockable nodes) and may materialize the query such that many non-entailments can easily be determined without a full entailment test (e.g., if a concept/role term does not occur in the completion graph or if fresh individuals that are related as specified by the role terms of the query yield a clash).

The ontologies used for the entailment checking experiments are depicted in Table 4, where the ontologies in the upper part contain more complex TBoxes such that consistency testing with tableau algorithms involves the generation of a significant amount of blockable nodes with non-tree-based connections to nominal nodes. Hence, query entailment checking must primarily be performed over blockable nodes and it is likely that blocking tests are required. In contrast, the lower part of Table 4 contains ontologies, where the completion graphs from the consistency checking process mainly consists of nodes that represent actual individuals in the knowledge base. Consequently, the existentially restricted parts are often less important for the query entailment checking. Note that we removed data properties and corresponding assertions for some ontologies (denoted with $_{\setminus\mathcal{D}}$) to make them consistent.

Table 4. Statistics and metrics for ontologies used for evaluating query entailment checking

Ontology	DL	TBox #A	TBox #C	TBox #P	ABox #I	ABox #CA	ABox #OPA	ABox #DPA
DMKB	$\mathcal{SROIQ}$	4,945	697	177	653	893	607	0
Family	$\mathcal{SROIQ(D)}$	317	61	87	405	1	1,089	433
Finance$_{\backslash\mathcal{D}}$	$\mathcal{ALCROIQ}$	1,391	323	247	2,466	2,466	343	0
GeoSkills$_{\backslash\mathcal{D}}$	$\mathcal{ALCHOIN}$	738	603	23	2,592	2,089	3,896	0
OBI	$\mathcal{SROIQ(D)}$	6,216	2,826	116	167	215	19	1
Wine	$\mathcal{SHOIN(D)}$	643	214	31	367	409	492	2
FMA3.1$_{\backslash\mathcal{D}}$	$\mathcal{ALCOIN}$	86,898	83,284	122	232,647	232,642	268,578	0
UOBM(1)	$\mathcal{SHOIN(D)}$	206	69	44	24,858	44,657	153,561	59,440

Table 5. Reasoning times for query entailment checks in seconds

Ontology	Preprocessing [s]	# Queries	Entailment checking [s] avg	med	min	max
DMKB	9.2	50	0.30	0.22	0.02	1.08
Family	24.2	50	180.32	$\geq$ 300.00	0.64	$\geq$ 300.00
Finance$_{\backslash\mathcal{D}}$	0.8	50	0.15	0.22	0.01	0.33
GeoSkills$_{\backslash\mathcal{D}}$	0.4	50	0.13	0.18	0.01	0.25
OBI	1.8	50	0.06	0.05	0.02	0.34
Wine	5.7	50	0.08	0.07	0.01	0.29
FMA3.1$_{\backslash\mathcal{D}}$	20.5	50	0.10	0.04	0.01	0.84
UOBM(1)	5.0	50	0.77	0.01	0.00	7.12

We evaluated each query separately, i.e., with a new instance of the reasoner, with a time limit of 5 minutes by using one core of a Dell PowerEdge R420 server with two Intel Xeon E5-2440 hexa core processors at 2.4 GHz (with Hyper-Threading) and 144 GB RAM under a 64bit Ubuntu 16.04.5 LTS.

The reasoning times for the query entailment checking experiments are depicted in Table 5. We separated the query independent pre-processing (i.e., loading, consistency checking, classification) from the statistical times for the actual query entailment checking. The median times (med) reveal that the entailment of most queries can be determined quite quickly. In fact, Table 6 shows more precisely how many queries can be answered within a certain time interval and, as one can see, 51% of the queries are handled in less than 0.1s and 90% in less than 1s. However, there are a few queries that lead to many propagations of variable mappings (e.g., UOBM) or require many analogous propagation blocking checks (e.g., Family) and, consequently, such queries require significantly more time. For the Family ontology, most of the queries (30) even reach the time limit, which seems due to complex roles that propagate variable mappings to most nominal as well as blockable nodes such that blocking tests become quite involved. However, the Family ontology is already quite difficult w.r.t. standard reasoning tasks, e.g., Pellet and HermiT cannot classify the ontology within 5 minutes. In addition, the implementation in Konclude currently tries to create and propagate all variable mappings at the same time, which is often quite useful for processing rule-based knowledge

Table 6. Query entailment checking distribution

Ontology	# Queries	Entailment checking intervals (in seconds)					
		$[0, 0.1)$	$[0.1, 1)$	$[1, 10)$	$[10, 100)$	$[100, 300)$	≥ 300
DMKB	50	2	47	1	0	0	0
Family	50	0	18	2	0	0	30
Finance$_{\setminus\mathcal{D}}$	50	20	30	0	0	0	0
GeoSkills$_{\setminus\mathcal{D}}$	50	22	28	0	0	0	0
OBI	50	48	2	0	0	0	0
Wine	50	44	6	0	0	0	0
FMA3.1$_{\setminus\mathcal{D}}$	50	36	14	0	0	0	0
UOBM(1)	50	31	13	6	0	0	0

(e.g., in form of nominal schema axioms) that applies to many individuals. In contrast, it could be more promising for query entailment checking to focus on a few variable mappings and try to complete the query with them. In particular, most of the propagation work is usually required for entailed queries since appropriate optimisations (e.g., pre-absorption of the query with ordinary concepts) are often able to detect non-entailment before the first variable mapping is created. Hence, a focus on a few variable mappings would probably often reveal entailment quickly, but would require a different propagation implementation/strategy than for the nominal schema axioms, which we are tried to avoid so far.

To further improve the performance, one could also use a representative propagation of variable mappings [27] for entailment checks and/or index more precisely which nodes constitute blocker candidates. However, it has to be considered that these queries are especially difficult and that this form of query entailment checking is only required in certain cases. In fact, it is usually possible to identify which individuals could be affected by the query (e.g., if the query contains answer variables) such that significantly smaller completion graphs can be constructed (with corresponding completion graph caching techniques).

6.2 Query Answering Evaluations and Comparisons

We compared the presented absorption-based query answering approach with OWL BGP, PAGOdA, and Pellet. Basically, we chose those query answering systems/approaches for our comparisons that are based on fully-fledged $\mathcal{SROIQ}$ reasoners, i.e., systems that are principally capable of answering conjunctive queries w.r.t. knowledge bases that are formulated with more expressive DLs. In the following, we describe the capabilities of the chosen query answering systems in more detail:

- **OWL BGP** [7] is an adapter for OWL API reasoners that enables the evaluation of SPARQL BGPs under the OWL 2 Direct Semantics Entailment Regime by reducing them to standard reasoning tasks for the reasoner, such as instance, satisfiability, and subsumption checking. In fact, OWL BGP even supports variables for concept and role terms, but only handles conjunctive queries with the restriction that all variables are interpreted as answer variables. Consequently, we do not expect that it

delivers complete results for queries with existential variables. The current version of OWL BGP is 0.1 and we used it with HermiT as the underlying reasoner since this is the "default configuration" and OWL BGP has special optimisations and query evaluation strategies that can be used in combination with HermiT (they can also be used for other reasoners if the corresponding interfaces are implemented). It is well-known that HermiT is a fully-fledged state-of-the-art reasoner for OWL 2, i.e., it completely supports the DL $\mathcal{SROIQ}$ with all OWL 2 datatypes, and is based on a variant of the tableau calculus, called hypertableau. To prepare the system for query answering, we also used the "default configuration" of the OWL BGP evaluation [7], which includes similar to Konclude the classification of concepts and roles. Realisation is only performed lazily while evaluating queries, based on lower and upper bound optimisations and sub-query bounds pruning [7].

- **PAGOdA** [31] can certainly be seen as the most advanced query answering system (for more expressive DLs) by delegating most of the work to a datalog reasoner/engine. In fact, the ontology is rewritten to lower and upper bound datalog programs and then the query is evaluated on both bounds with the datalog reasoner. If there are differences in the answers, then a fully-fledged DL reasoner is used to verify the additional answers of the upper bound. PAGOdA also implements several optimisations to reduce the work of the DL reasoner. For example, it tries to extract a subset of the ontology that is sufficient for the DL reasoner to verify the remaining answers. In addition, summarisation techniques are used and dependencies of answers are analysed to minimize the requests for the DL reasoner. The support of existential variables depends on the capabilities of the DL reasoner since it can be required to fall back to a fully-fledged query answering technique. Nevertheless, even if the used DL reasoner only supports instance checking/retrieval, then conjunctive queries with existential variables can be answered to some extent by using a so-called combined approach for less expressive DLs, where first the datalog program is (query independently) augmented by new constants that represent existentially restricted individuals and, then, unsound answers of the evaluation of the query are "filtrated" out. We used PAGOdA in the "default configuration" with RDFox as the underlying datalog engine and HermiT as the fully-fledged DL reasoner.[2] As preparation step, PAGOdA is rewriting the ontology into several datalog programs, with which then the given datasets are materialised by the underlying datalog reasoner.
- **Pellet** [24] is another fully-fledged reasoner for the DL $\mathcal{SROIQ}$ with integrated support for conjunctive queries. In particular, Pellet uses the Jena framework for handling and evaluating SPARQL queries and supports existential variables (to some extent) via the rolling-up technique. Since the Pellet reasoner cannot directly be accessed within the Jena framework when evaluating SPARQL queries, we did not implement the prepare command. Consequently, we interpret the fastest query

[2] We used the "JAIR 2015" version of PAGOdA for the evaluation since the last version from the repository caused some errors and unexpected results. Since the original PAGOdA repository has been deleted, we uploaded a copy at `https://github.com/andreas-steigmiller/PAGOdA`.

answering response of Pellet as preparation time. We used the latest version (2.3.1) of Pellet from the official Github repository.[3]

For evaluating a query w.r.t. an ontology for a query answering system, we wrote for each system a simple SPARQL HTTP wrapper server such that the processing of the ontologies and queries can uniformly be controlled for the different systems. In fact, we use commands of the SPARQL Update protocol to initiate the loading of the ontologies and datasets. Subsequently we send a "prepare command" such that the query answering system can preprocess the ontology in order to be ready for query answering. Note that it is not clear for all systems how such a "prepare command" has to be translated into corresponding method calls of the reasoner by the wrapper and, thus, we only implemented it for some of the systems. For the remaining query answering systems, we interpreted the fastest query answering response as preprocessing/preparation time. For Konclude, the "prepare command" leads to the syntactical preprocessing of the ontology (e.g., constructing the internal representation, absorption, etc.) and, subsequently, to a consistency check as well as the classification of all atomic concepts and roles (since the concept and role hierarchies are used by the on-demand realisation service).

For evaluating and comparing the systems, we used, on the one hand, the well-known ontologies and queries from the PAGOdA evaluation (see Section 6.2.1), and, on the other hand, we generated and evaluated many random queries for ontologies of the ORE2015 dataset (see Section 6.2.2).

6.2.1 Comparison of Query Answering Systems for Well-Known Benchmarks

For a basic comparison with other query answering systems, we used the ontologies and (non-trivial) queries from the PAGOdA evaluation [31]. Table 7 shows the most interesting statistics and metrics for the ontologies from the PAGOdA evaluation, which are mostly well-known benchmark ontologies and include data of several real-world application scenarios. 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., [20]). The ontologies from the PAGOdA evaluation are separated into two files. The first file contains the core ontology with all the TBox as well as RBox axioms and possibly a few assertions for the individuals that are used as nominals, whereas the second file represents the actual dataset and provides all the remaining ABox assertions. Note that for most of these ontologies, there exist several different datasets. In fact, some of the ontologies are synthetic benchmarks, where arbitrarily big datasets can be obtained with generators and for some of the very big ontologies sampling techniques have been used to obtain smaller datasets. Since these expressive ontologies easily become problematic for several reasoning systems, we only used the smallest versions of the available datasets. The most important aspects of the ontologies and their datasets can be summarised as follows:

- **LUBM** [9] and **UOBM** [16] are well-known benchmark ontologies about universities, enrolled students, and their activities. Generators can be used to obtain arbitrarily big synthetic datasets, but we used the smallest ones with only one university, which we denoted with LUBM(1) and UOBM(1). In addition to the provided

[3] `https://github.com/stardog-union/pellet`

Table 7. Statistics and metrics of the ontologies from the PAGOdA evaluation for comparing query answering systems

Ontology	DL	TBox			ABox			
		#A	#C	#P	#I	#CA	#OPA	#DPA
$\text{ChEMBL}_{1\%}$	$\mathcal{SRIQ}(\mathcal{D})$	3,171	1,706	411	2,891,835	2,028,496	429,139	427,261
FLY	$\mathcal{SRI}$	20,715	7,533	24	1,606	803	803	0
LUBM(1)	$\mathcal{ALEHI}^{+}(\mathcal{D})$	93	43	32	17,174	18,128	49,336	33,079
$\text{Reactome}_{10\%}$	$\mathcal{SHIN}(\mathcal{D})$	600	69	96	628,759	628,759	276,740	409,141
$\text{Uniprot}_{1\%}$	$\mathcal{ALCHOIQ}(\mathcal{D})$	608	164	135	407,638	407,714	514,538	190,189
UOBM(1)	$\mathcal{SHIN}(\mathcal{D})$	246	113	44	24,858	44,657	153,561	59,440

queries for LUBM (14) and UOBM (15), we also used the more challenging queries introduced by Zhou et al. [31] that further contain some existential variables. As one can see from the object property/role assertions (see #OPA in Table 7), the individuals of LUBM(1) and UOBM(1) are intensively connected, but the TBoxes are rather simple, i.e., implying only few anonymous individuals.

- **FLY** is an ontology about the anatomy of the brain of the Drosophila (a certain genus of flies) and is used in Virtual Fly Brain browser application.[4] The ABox contains only a few trivial concept and role assertions, but the TBox includes many axioms with existential restrictions, which makes query answering, in combination of the complex roles of the ontology, quite challenging. We used the 6 realistic queries that were provided by the developers for the PAGOdA evaluation [31].
- **ChEMBL**, **Reactome** and **Uniprot** are ontologies provided by European Bioinformatics Institute[5] (EBI), which describe bioactive molecules with drug-like properties, biological pathways, and protein sequences and functional informations, respectively. We used those samples generated for the PAGOdA evaluation [31] that contain only 1%, 10%, and 1% of the original datasets, respectively. The datasets contain many, but only sparely connected individuals. Nevertheless, query answering is challenging for these ontologies due to the large amount of non-determinism that is caused by TBox axioms (especially from disjunctions). We also used those (non-trivial) example queries that were collected from the EBI website for the PAGOdA evaluation [31].

The evaluation was carried out on a Dell PowerEdge R420 server running with two Intel Xeon E5-2440 hexa core processors at 2.4 GHz (with Hyper-Threading) and 144 GB RAM under a 64bit Ubuntu 16.04.5 LTS. As far it has been configurable, we used only one core for each reasoner to facilitate the comparison. 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 8. By considering the accumulated times over all ontologies (last row), one can say that Konclude outperforms the other systems for the evaluated ontologies and queries. In fact,

[4] `http://www.virtualflybrain.org/site/vfb_site/overview.htm`
[5] `https://www.ebi.ac.uk/`

Table 8. Query answering times in seconds for the queries and ontologies from the PAGOdA evaluation (without preprocessing of the ontologies)

Ontology	# Queries	Konclude	OWL BGP	PAGOdA	Pellet
$ChEMBL_{1\%}$	6	0.3	1800.0	0.7	17.7
FLY	11	4.7	3300.0	20.3	3300.0
LUBM(1)	35	1.2	16.8	5.4	11.2
$Reactome_{10\%}$	7	0.4	2100.0	14.4	33.8
$Uniprot_{1\%}$	13	0.3	3900.0	2.7	307.8
UOBM(1)	20	6.6	6000.0	22.7	972.7
ALL	92	13.6	17116.8	66.1	4643.2

Konclude is faster than the other systems for all ontologies except $ChEMBL_{1\%}$, where it requires as much time as PAGOdA to answer the 6 complex queries. Although PAGOdA can answer all queries for all evaluated ontologies, it is clearly slower than Konclude for $Reactome_{10\%}$ and the FLY ontology. The latter may be explainable due to the complex TBox of the FLY ontology with many existential restrictions and an RBox with several non-trivial complex roles. Translating such "complex" ontologies into datalog programs and managing potentially many new constants for existentially qualified concepts can be more involved. In addition, 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.

By only timing out for the FLY queries and one UOBM(1) query, Pellet also seems to be a quite robust query answering system, but it requires significantly more time in several cases. Note that the depicted response times for Pellet are even a little bit better than they actually are since we subtracted the fastest response for each ontology to eliminate the preprocessing times. If we consider the queries/ontologies, where Pellet has its best response times (e.g., $ChEMBL_{1\%}$, LUBM(1), $Reactome_{10\%}$), then we can still observe that they are significantly slower than the answering times of Konclude. On the one hand, we assume that this is due to the different query answering approach, which is deeper integrated into the reasoning system for Konclude and, hence, it can better utilise the internal data structures and can profit more from corresponding (reduction) optimisations (see, e.g., Section 5). On the other hand, some of the performance differences could stem from the overall reasoning system and the integrated optimisations. In particular, Konclude is implemented in C++, whereas Pellet is based on Java. Moreover, Konclude is designed as a high performance reasoner with a wide range of optimisations that make reasoning very fast, whereas Pellet also focuses on extensions to reasoning (e.g., explanation services).

OWL BGP is only able to handle the LUBM(1) ontology with the corresponding queries. It is also not very surprising that OWL BGP requires significantly more time than the other systems for the LUBM queries since OWL BGP is merely an adapter and has to use the official OWL API reasoner interface to resolve the query answers, i.e., the query answering technique is not really deeply integrated into the reasoning system. In addition, it has to be noted that OWL BGP does not have a particular focus on conjunctive query answering. For the remaining ontologies, it seems that they are

Table 9. Total processing times for the queries and ontologies from the PAGOdA evaluation in seconds (i.e., ontology loading and preprocessing is included as well as the query answering time)

Ontology	# Queries	Konclude	OWL BGP	PAGOdA	Pellet
$\text{ChEMBL}_{1\%}$	6	233.4	1800.0	479.7	717.5
FLY	11	28.0	3300.0	408.3	3300.0
LUBM(1)	35	38.7	520.1	121.4	134.6
$\text{Reactome}_{10\%}$	7	174.0	2100.0	1103.0	688.8
$\text{Uniprot}_{1\%}$	13	255.4	3900.0	511.6	1223.7
UOBM(1)	20	96.0	6000.0	304.5	1088.5
ALL	92	825.4	17620.1	2928.4	7153.1

problematic for HermiT since often not even preprocessing (i.e., consistency checking and classification) can be finished within the time limit.

The UOBM queries are the most difficult ones for Konclude. In fact, Konclude requests, for some of the UOBM queries, the realisation of concepts that are defined with number restrictions. For the resulting instance tests, Konclude has to perform a lot of merging operations such that the processing of the completion graph has to be extended to many individuals. Since the realisation results are typically cached, they can, in principle, be reused such that subsequent queries, which rely on the same concepts, can be answered much faster. Of course, this also holds for most of the other query answering approaches, i.e., the query answering times are usually getting better the more queries are processed.

Table 9 further shows the total processing times, i.e., loading and preprocessing of the ontology is included for each query. Again, it is observable that Konclude has the least processing time for each ontology, even when compared with PAGOdA. Consequently, Konclude is not doing more preprocessing or, at least, it is much more efficient. One can also observe that PAGOdA indeed requires much more preprocessing time for ontologies with complex TBoxes and RBoxes, such as the FLY ontology, where Konclude is almost 15 times faster than PAGOdA, whereas PAGOdA only requires two or three times the time of Konclude for the other ontologies. Although the underlying datalog engine of PAGOdA is also implemented in C++ and the datasets are directly and very efficiently loaded by the datalog engine (i.e., PAGOdA only delegated the loading command for the datasets to the datalog engine), PAGOdA manages and requests the materialisation of several instances of the ontologies/datasets, which may explain some of the processing time differences between Konclude and PAGOdA.

We further evaluated a version of Konclude[6] that interprets all non-selected SPARQL variables as existential variables. In fact, most queries from the PAGOdA evaluation have only a few existential variables and often only in tree-shaped query parts such that they can also be handled by some restricted query answering systems. This version of Konclude required only a few additional seconds (21.7s) compared to default version

[6] The extended version of Konclude is based on slightly newer sources (v0.6.2-927), which include a few changes such that the performance for the PAGOdA queries/ontologies is generally a little bit weaker. Most notably, a full query materialisation has been integrated and query answers are computed incrementally.

Table 10. Statistics w.r.t. the generated queries for the ORE2015 ontologies

Value	avg	med	min	max
# Variables	5.43	5	3	20
# Selected variables	2.36	2	1	10
# Cycles	3.34	3	1	25
# Role terms	6.82	6	3	47
# Concept terms	5.78	5	0	43
# Query terms	16.6	12	4	86

of Konclude (13.6s), but it computed additional answers for 14 queries (i.e., 15 % of all queries). More precisely, additional answers were found for 1 Uniprot query, 4 FLY queries, and for 9 of the additional LUBM queries that have been introduced/suggested by the developers of the PAGOdA system.

6.2.2 Comparison of Query Answering Systems for Randomly Generated Queries

In order to get a more comprehensive evaluation, we generated 4, 502 cyclic queries for the ontologies of the ORE2015 dataset [20]. We focused on cyclic queries since they do prevent reduction techniques (such as rolling up) and, hence, can be considered as more challenging. It is worth pointing out that all ORE2015 ontologies are only in the OWL 2 DL profile, i.e., they use a combination of language features such that they are not in the OWL 2 EL, nor in the OWL 2 RL, and also not in the OWL 2 QL profile. For the query generation, we used the fully expanded and clash-free completion graphs generated by Konclude in the course of checking consistency for these ontologies. Since Konclude does not always construct complete and connected completion graphs (e.g., individuals are ignored if their restrictions are trivially satisfied), we deactivated corresponding optimisations. For each completion graph, we tried to extract 10 queries by randomly picking one node representing an individual and by analysing the local neighbourhood. More precisely, we visited new neighbour nodes that also represent individuals in a breath-first based manner and saved the paths until a certain depth was reached. If a node was visited several times, then we traced back all paths and generated a query (with possibly several cyclic parts) by using for each node a unique variable name and by building query terms with randomly picked concepts and roles from the labels of the involved nodes and edges. Note that some of the queries may not have an answer if the picked concepts or roles were derived non-deterministically. For obtaining SPARQL queries, we randomly picked up to 10 variables for the SELECT clauses. For each completion graph, we limited the query generation attempts to 1, 000 with at most 10 queries per ontology, which resulted in overall 4, 502 queries for 511 ontologies (in average 8 queries per ontology).

Note that 1, 019 of the 1, 920 ontologies in the ORE2015 dataset do not contain any individuals and, thus, no queries were generated for these ontologies. For the remaining 300 ontologies, Konclude (with some optimisations deactivated) could either not construct a fully expanded and clash-free completion graph within the time limit of 5 minutes (e.g., if the ontology is inconsistent or too "difficult") or the query generation failed to find any cyclic connection between individuals. Table 10 shows the most im-

Table 11. Statistics w.r.t. the ontologies of generated queries from the ORE2015 dataset

Value	avg	med	min	max
# Individuals	9,733.2	1,155	6	658,369
# Classes	3,099.3	841	8	150,591
# Object properties	217.7	118	3	11,312
# Data properties	24.8	22	0	2,275
# Axioms	38,765.2	10,191	241	1,205,500
# Assertions	30,482.7	1,695	5	1,202,310

portant statistics w.r.t. the generated queries for the ontologies of the ORE2015 dataset. In contrast, Table 11 shows statistics for those ontologies of the ORE2015 dataset for which queries have been generated. Note that each ontology is weighted by the number of queries for this ontology for computing the average and median values of Table 11.

In contrast to the evaluation of the PAGOdA ontologies and queries, we evaluated the generated ORE2015 queries on a Dell PowerEdge R720 server running with two Intel Xeon E5-2680 hexa core processors at 2.66 GHz (with Hyper-Threading) and 512 GB RAM in a containerized (lxd) environment under a 64bit Ubuntu 18.04 LTS. We configured Konclude (v0.6.2-927) to use only one CPU core and limited the memory usage to 100 GB RAM. We evaluated each query with a fresh instance of the reasoner and a time limit of 5 minutes. Again, if the preprocessing did not finish within the time limit, then we also interpreted the query answering task as a timeout, i.e., as 5 minutes. Since some of the queries have more than several hundred million answers (requiring several hundred of GBytes in the uncompressed SPARQL XML result serialization), we stopped the evaluation process as soon as the SPARQL client received 100 GBytes of data and interpreted it as completed. Moreover, we wrote the query answers to files only if they did not exceed 200 MBytes. Consequently, we could not analyse and compare results for queries that exceeded the limit.

The accumulated query answering times over all 4,502 generated queries of the ORE2015 dataset are 14,412s for Konclude, 190,435s for OWL BGP, 77,143s for PAGOdA, and 231,057s Pellet (cf. Figure 9). Since we did not find a simple way to separate the preprocessing times for Pellet, we subtracted for each query the fastest query answering response for that ontology. As a result, the answering time for several queries is interpreted as 0 or close to 0 (if queries behave similar) and, thus, the actual performance of Pellet is likely to be worse than depicted. This also results in the unusual curve for Pellet on right-hand side graph of Figure 9, where the query answering times are separately sorted for each reasoner, i.e., the fastest response for each reasoner is on the left-hand side, whereas the most “difficult” query provide the values for the right-hand side for that reasoner. As it can be observed from the graph on the right-hand side of Figure 9, Konclude has the best performance for most queries. The total processing times (i.e., loading and preprocessing included) are 24,343s for Konclude, 236,954s for OWL BGP, 130,752s for PAGOdA, and 262,279s for Pellet. Konclude reached the time limit of 5 minutes for 27 queries, OWL BGP for 598, Pellet for 730, and PAGOdA for 235 queries.

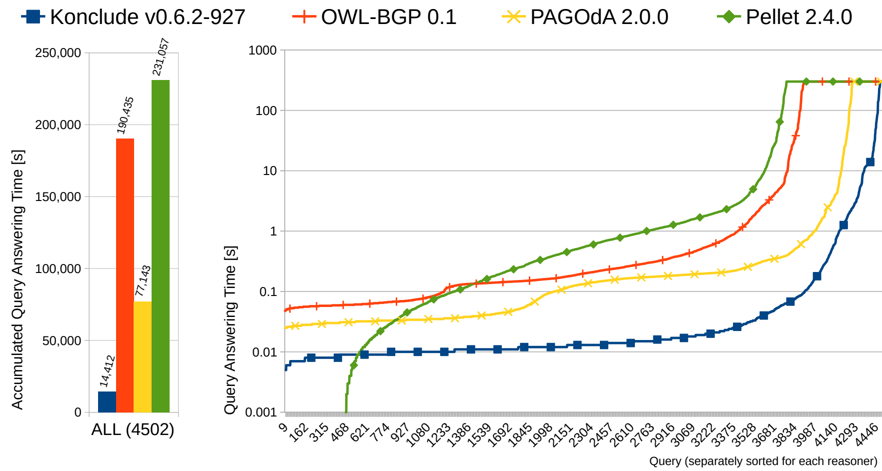

Fig. 9. The query answering times over the cyclic ORE2015 queries for state-of-the-art query answering systems (on left-hand side accumulated over all queries; on right-hand side separately sorted for each reasoner)

Unfortunately, several queries were not processed successfully by a few reasoners, which manipulates the evaluation and the comparison to some extent. In fact, OWL BGP failed to process 1,300 queries, PAGOdA 340, and Pellet 30 queries. If a query resulted in a processing error/failure, then we interpreted the time between sending the query and receiving the error as processing/response time. Konclude processed all queries without producing/reporting errors, but it has to be noted that we could only generate queries for those ontologies for which Konclude could already construct a fully expanded and clash-free completion graph. However, Konclude is regularly tested with large amounts of ontologies, which helps in minimizing processing errors. In fact, consistency checking resulted only for 4 ontologies of the ORE2015 dataset in processing errors for the evaluated version of Konclude, which, for example, were caused from invalid data literals, unsupported parts of SWRL rules, and by reaching the memory limit. The errors for the other reasoners have a range of different causes and we only summarize the most relevant ones in the following. The OWL BGP adapter based on HermiT refused to process many ontologies that use an AnonymousIndividual expression as nominal. In contrast, PAGOdA did not always succeed in preparing and extracting the relevant ontology module for the fully-fledged reasoner (producing conversion/casting errors) and the SPARQL parser of the underlying datalog engine (RDFox) failed to parse some queries due to encoding problems (we used parts of the individual names, possibly including some non-ASCII characters, for getting unique variable names). Pellet also stopped the processing for some invalid/unsupported data literals.

For a surprisingly large amount of queries (1677), at least one reasoner computed a different result, i.e., a reasoner found an answer that was not computed by the other systems (since the cardinality of the answers is not reported by PAGOdA, we also ignored it for the result comparison). If the results coincide for all reasoners but one, then it may

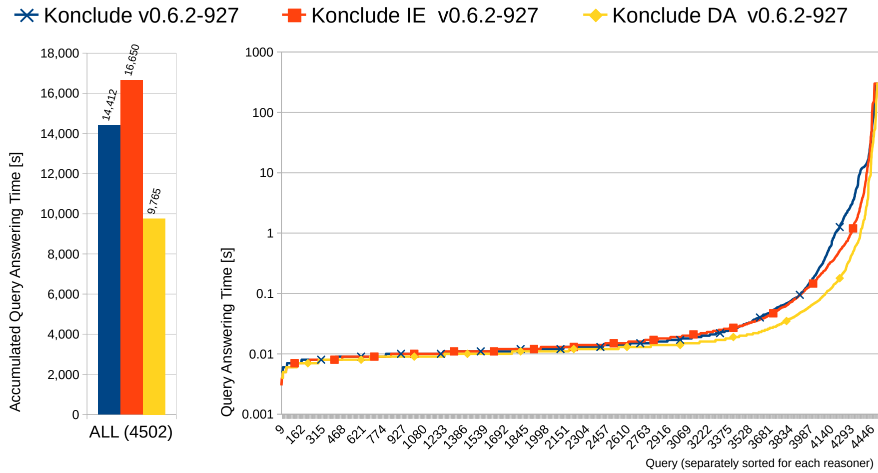

Fig. 10. The query answering times over the cyclic ORE2015 queries for different versions of Konclude (on left-hand side accumulated over all queries; on right-hand side separately sorted for each reasoner)

indicate that the reasoner with the additional or missing answers computed a wrong or invalid result. This seems to be the case for Konclude for 0, for OWL BGP for 91, for Pellet for 15, and for PAGOdA for 526 queries. The latter may indicate that PAGOdA does not yet fully support all language features of OWL 2.

Note that the generated ORE2015 queries do not have any existential variables and, hence, can simply be evaluated by realizing (parts of) the ontology and by appropriately joining instances for role and concepts terms. Nevertheless, Konclude may propagate some variable mapping through the completion graph in order to reduce the realisation tasks (as discussed in Section 5.2). To measure the performance impact of different query evaluation restrictions, we further evaluated the following versions of Konclude:

- **Konclude IE** (for **I**nterpreting **E**xistentially) interprets non-selected SPARQL variables as existential variables. This usually forces Konclude to absorb query parts and to create and propagate variable mappings through the completion graph, as presented in the previous sections. As a result, Konclude may calculate/find some additional answers, but it can be more costly in term of performance than simply using realisation results. Note that the total amount of answers can also be less since existential variables are not considered for cardinalities.
- **Konclude DA** (for **D**istinct **A**nswering) computes only distinct answers, i.e., the cardinalities of the answers are ignored. As a consequence, much smaller SPARQL results have to be generated for most queries since usually only a few variables are used in the SELECT clause.

The accumulated query answering times over the generated ORE2015 queries are $16,650$s for Konclude IE and $9,764$s for Konclude DA (cf. Figure 10). Interestingly, Konclude DA is quite a bit faster than the default version of Konclude, which also rep-

resents intermediate results distinctly by mapping bindings to a number that represents the cardinality. If the cardinalities are, however, extremely large, then the serialisation of the corresponding answers starts to require a significant amount of time (although it is quite efficiently realised in Konclude by first writing in a temporary buffer that is subsequently just multiple times written/copied). For several queries, even the transfer of the serialised results over the loopback network interface becomes the bottleneck (the used HTTP connections did, however, not utilise compression techniques). It can be questioned whether such queries are realistic or make a lot of sense, but a more efficient SPARQL result format would certainly be useful (e.g., by including the cardinality for each answer instead of repeating it several times). Clearly, the generation of random queries can result in some redundant variables that may lead to such large cardinalities. On the other hand, many ontologies of the ORE2015 dataset contain a lot of same individuals due to SameAs axioms from integrated ontology alignments. Although Konclude mostly handles each same individual group with one representative individual for reasoning, each combination must separately be written as an answer, i.e., it easily leads to a combinational explosion. Again, an adapted SPARQL result format, where a binding constitutes a synset of same individuals, could be more efficient for the communication with query answering systems that may derive that individuals are the same. Nevertheless, Konclude IE only requires a little bit of additional time (compared to the default version), but it computed additional results for 218 queries, even though the queries are generated in a way such that they focus on the known individuals of the ontology and not on the existential/implied part. It can further be observed that some queries become easier and can faster be answered by Konclude IE than for the default version, which mainly seems to be caused by the fact that the answers are also returned distinctly since cardinalities are not considered for existential variables. In addition, some existential variables can be eliminated since they do not have to be considered for cardinalities and their restrictions are already represented with other variables and query terms. However, Konclude IE failed to handle a few queries in contrast to Konclude since a large amount of variable mappings had to be propagated through the completion graph such that Konclude IE reached the memory limit. This may be addressable by splitting the propagation for the binding candidates extraction into several separate completion graph construction tasks instead of trying to handle it in a one-pass manner. At the same time, such a split up could be quite suitable for parallelization.

6.3 Discussion

Clearly, the comparisons with other systems do not only evaluate the query answering approach, but also the underlying reasoning systems. On the one hand, this cannot completely be avoided since certain query answering approaches require corresponding optimisations and techniques in the reasoning system. For example, our absorption-based query answering approach requires a tableau-based reasoner that is principally capable of managing and processing binary inclusion axioms. Moreover, the reasoning system should incorporate state-of-the-art absorption or equivalent techniques (see, e.g., [17,26]) such that non-determinism is avoided as much as possible and accompanying optimisations (see Section 4.3 and 5) work well. On the other hand, implementing query answering in form of an adapter prevents the utilisation of reasoner internals

and often comes with an additional overhead, as also indicated by our evaluation. It would be interesting to include versions of OWL BGP and PAGOdA that are based on Konclude (instead of HermiT) in our evaluation since this would facilitate the comparison of the query answering approach. Unfortunately, Konclude does not yet implement the OWL API reasoner interface directly and, hence, the communication (e.g., via OWLlink) would result in a significant overhead, which does not seem a feasible option at the moment. Nevertheless, the comparison indicates that the presented query answering approach can indeed be implemented such that it works quite well for real-world ontologies and queries.

7 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 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 enables 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 other systems. Moreover, our evaluation shows that queries with existential variables indeed result in additional answers for many real-world ontologies, which can be seen as an indication that the more exhaustive query answering can be quite relevant in practice.

8 Acknowledgements

This research was funded by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG), project number 330492673.

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.: Querying Description Logic Knowledge Bases. Ph.D. thesis, University of Manchester, United Kingdom (2007)
5. Glimm, B., Horrocks, I., Sattler, U.: Conjunctive query answering for description logics with transitive roles. In: Proc. 19th Int. Workshop on Description Logics (DL'06). CEUR WS Proceedings, vol. 189, pp. 3–14. CEUR (2006)
6. 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)
7. 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)
8. 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)
9. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. J. of Web Semantics **3**(2-3), 158–182 (2005)
10. 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)
11. 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)
12. 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)
13. 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)
14. Kollia, I., Glimm, B.: Optimizing SPARQL query answering over OWL ontologies. J. of Artificial Intelligence Research **48**, 253–303 (2013)
15. Levy, A.Y., Rousset, M.C.: Combining Horn rules and description logics in carin. Artificial Intelligence **104**(1–2), 165–209 (1998)
16. Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y., Liu, S.: Towards a complete OWL ontology benchmark. In: Proc. 3rd European Semantic Web Conf. (ESWC'06). LNCS, vol. 4011, pp. 125–139. Springer (2006)
17. Motik, B., Shearer, R., Horrocks, I.: Hypertableau reasoning for description logics. J. of Artificial Intelligence Research **36**, 165–228 (2009)
18. 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)
19. 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)
20. 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)
21. 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)
22. Simančík, F.: Elimination of complex RIAs without automata. In: Proc. 25th Int. Workshop on Description Logics (DL'12). vol. 846. CEUR (2012)
23. Sirin, E., Cuenca Grau, B., Parsia, B.: From wine to water: Optimizing description logic reasoning for nominals. In: Proc. 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'06). pp. 90–99. AAAI Press (2006)

24. 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)
25. Steigmiller, A., Glimm, B.: Absorption-Based Query Answering for Expressive Description Logics : Evaluation Data (2019). https://doi.org/10.5281/zenodo.3266159, `https://doi.org/10.5281/zenodo.3266159`
26. Steigmiller, A., Glimm, B., Liebig, T.: Optimised absorption for expressive description logics. In: Proc. 27th Int. Workshop on Description Logics (DL'14). vol. 1193. CEUR (2014)
27. Steigmiller, A., Glimm, B., Liebig, T.: Reasoning with nominal schemas through absorption. J. of Automated Reasoning **53**(4), 351–405 (2014)
28. Steigmiller, A., Glimm, B., Liebig, T.: Completion graph caching for expressive description logics. In: Proc. 28th Int. Workshop on Description Logics (DL'15) (2015)
29. Steigmiller, A., Liebig, T., Glimm, B.: Konclude: system description. J. of Web Semantics **27**(1) (2014)
30. Stoilos, G., Stamou, G.: Hybrid query answering over OWL ontologies. In: Proc. 21st European Conf. on Artificial Intelligence (ECAI'14). pp. 855–860 (2014)
31. 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)