I N F S Y S
R E S E A R C H
R E P O R T

INSTITUT FÜR INFORMATIONSSYSTEME

ARBEITSBEREICH WISSENSBASIERTE SYSTEME

CONJUNCTIVE QUERY ANSWERING IN THE DESCRIPTION LOGIC $\mathcal{SH}$ USING KNOTS

Thomas Eiter Magdalena Ortiz Mantas Šimkus

INFSYS RESEARCH REPORT 1843-09-03

SEPTEMBER 2009

Institut für Informationssysteme
AB Wissensbasierte Systeme
Technische Universität Wien
Favoritenstrassße 9-11
A-1040 Wien, Austria
Tel: +43-1-58801-18405
Fax: +43-1-58801-18493
sek@kr.tuwien.ac.at
www.kr.tuwien.ac.at

INFSYS RESEARCH REPORT

INFSYS RESEARCH REPORT 1843-09-03 , SEPTEMBER 2009

CONJUNCTIVE QUERY ANSWERING IN THE DESCRIPTION LOGIC $\mathcal{SH}$ USING KNOTS

Thomas Eiter,[1] Magdalena Ortiz,[2] Mantas Šimkus[3]

Abstract. Answering conjunctive queries (CQs) has been recognized as an important task for the widening use of Description Logics (DLs) in a number of applications. The problem has been studied by many authors, who developed a number of different techniques for its solution. We present a novel method for CQ answering based on knots, which are schematic subtrees of depth at most one. The method yields an algorithm for CQ answering in the DL $\mathcal{SH}$ which handles CQs with distinguished (i.e., output) variables in a direct manner. It proceeds by first compiling the knowledge base into a set of knots, and then constructing a set of simple knowledge bases, which contain only assertional data, over which a given query is answered. Notably, the knot compilation can be reused for varying queries and is amenable to an implementation in disjunctive Datalog. The algorithm works in double exponential time in general but in single exponential time under various restrictions on the occurrence of transitive roles in queries, including CQ answering in the DL $\mathcal{ALCH}$. The results are worst-case optimal, given that CQ answering is 2EXPTIME-complete for $\mathcal{SH}$ and EXPTIME-hard already for the core expressive DL $\mathcal{ALC}$. In particular, the result for $\mathcal{ALCH}$ reconfirms Lutz's result that adding inverse roles to $\mathcal{ALC}$ causes an exponential jump in complexity, while adding role hierarchies does not. Furthermore, a nondeterministic version of our algorithm runs in CONP under data complexity, which is worst-case optimal in this setting as well.

[1]Institute of Information Systems, Vienna University of Technology. E-mail: eiter@kr.tuwien.ac.at.
[2]Institute of Information Systems, Vienna University of Technology. E-mail: ortiz@kr.tuwien.ac.at.
[3]Institute of Information Systems, Vienna University of Technology. E-mail: simkus@kr.tuwien.ac.at.

Acknowledgements: This work has been partially supported by the Austrian Science Fund (FWF) grant P20840, the Mexican National Council for Science and Technology (CONACYT) grant 187697, and the EU project OntoRule (IST-2009-231875).

Some results of this report have appeared, in preliminary form, in conference papers: in *Proc. AAAI '08* [31] and *Proc. DL'08* [30].

Contents

1 Introduction

In the last years, Description Logics (DLs) have increasingly received attention as a tool for representing domain models in various application areas. Among these areas are data and information integration, peer-to-peer data management, ontology-based data access, and the Semantic Web. The widening use of DLs also raised the need for reasoning services beyond traditional services like satisfiability testing, determining subsumption relationships, and instance checking. In particular, answering conjunctive queries (CQs) over knowledge bases in DLs has been recognized as a primary such task. Indeed, CQs allow to join pieces of information and are at the heart of database query languages like SQL; for example, if a relation $emp(e, d)$ stores data about employees e that work in departments d, and a relation $dept(d, a)$ stores data about departments d and their addresses a, then the CQ $emp(x, y), dept(y, z)$ joins the information in the relations and yields for each employee x her work address z. Thus, supporting CQs over DL knowledge bases is, for instance, important for the use of DLs as a formalism for rich data models.

Driven by this need, the problem has been studied in many papers, including [29, 19, 13, 14, 22, 6, 5, 2, 20, 28, 33], and a number of results have been derived for a range of DLs. In DLs that extend the core expressive DL $\mathcal{ALC}$, like $\mathcal{SHIQ}$ and $\mathcal{SRIQ}$ (which correspond to the Web Ontology Language standard of the W3C) and $\mathcal{DLR}$, answering CQs is at least EXPTIME-hard as it subsumes the satisfiability problem of $\mathcal{ALC}$ knowledge bases, which is well-known to be EXPTIME-complete. However, the problem is harder for many DLs; e.g., it is 2EXPTIME-hard for all DLs containing $\mathcal{ALCI}$ [22] or $\mathcal{SH}$ [10]. On the other hand, 2EXPTIME upper bounds are known for these logics and the embracing DL $\mathcal{SHIQ}$, cf. [6, 2, 19, 13], which are thus tight. However, for other extensions of $\mathcal{ALC}$, like $\mathcal{ALCH}$ and $\mathcal{ALCHQ}$, answering CQs is still feasible in single exponential time [22, 23, 31],[1] and thus the problem is not more expensive than the satisfiability problem in these logics.

To design CQ answering algorithms in expressive DLs, various approaches have been used; they range from incorporating the query into the knowledge base [6, 37, 13, 14] over adapting tableaux procedures [21, 29, 28, 33] and applying resolution-based techniques [19] to automata-based algorithms [2, 20]. In this paper, we consider a different method, which is based on the knot technique. Knots are schematic trees of depth at most one that occur in the forest-shaped models of a DL knowledge base. They have been introduced in the context of non-monotonic logic programming for $\mathbb{FDNC}$ programs [36] and can be seen as a special instance of mosaics in modal logic [11, 9].

The main result of this paper is a novel algorithm for answering CQs over $\mathcal{SH}$ knowledge bases. It extends a similar algorithm for $\mathcal{ALCH}$ presented in [31] and works in double exponential time in general, but in single exponential time for queries from large fragments of $\mathcal{SH}$ including $\mathcal{ALCH}$, which emerge by restricting the occurrence of transitive roles in queries. More precisely, the algorithm has the following features:

- It is worst-case optimal for arbitrary CQs over $\mathcal{SH}$ as well as for CQs with restricted occurrence of transitive roles, e.g. CQs containing only few (bounded by a constant many) atoms involving transitive roles; this restriction seems not to be severe in practice. In particular, the algorithm is thus worst-case optimal also for answering CQs over $\mathcal{ALCH}$ knowledge bases. This contrasts with several other algorithms for CQ answering in $\mathcal{SH}$ which either do not have a double exponential upper bound, or need double exponential time already for fragments like $\mathcal{ALCH}$ (see Section 7 for more details).

- Different from other algorithms, it handles CQs with answer variables (alias distinguished or output variables) in a direct manner, rather than reducing such queries to ground (Boolean) CQs. In our

[1] Lutz announced his result in [22], and more details were given later in [23].

example above, the variables x and z, which intuitively return an employee with her work address, are answer variables. Reducing the query to ground CQs is achieved by binding x and z to all possible employees and addresses, which might be rather inefficient (e.g., if it appears that each employee has a unique work address).

- The algorithm provides a modular *knowledge compilation* of the DL knowledge base, which allows for the reuse of intermediate results. This is because it compiles first a DL n knowledge base into a set of knots, constructs then query answering tables from this set and the input query, and finally collects the query answers from the tables using the data part (in DL jargon, the ABox) of the knowledge base. The result of the first step may be reused for follow-up queries, i.e., only the query answering tables need to be constructed and the query answers collected. For queries of small size (bounded by a constant), the table construction is feasible in polynomial time in the size of the knot set, and collecting the query answers is feasible in CONP (viewed as a decision problem); for a fixed ABox, the latter is feasible in polynomial time. This is particularly useful for evaluating many such queries over a rather static knowledge base.

- Similarly, for a fixed query and a DL knowledge base where the terminological component (the TBox) is fixed but the ABox may change, i.e., in the *data complexity* setting, a non-deterministic version of our algorithm runs in polynomial time. This means that the algorithm is also worst-case optimal under data complexity, as answering CQs is known to be CONP-complete for a wide range of DLs from $\mathcal{AL}$ to $\mathcal{SHIQ}$ (cf. [5, 16, 28]).

- Finally, the compiled knowledge can be expressed as a disjunctive Datalog program (alternatively, a Datalog program with unstratified negation), which is evaluated over an enhanced ABox. The program can be designed to evaluate also non-ground queries, i.e, with answer variables directly. A Datalog encoding may make the algorithm more amenable for efficient implementation than some of the previous automata- or tableaux-based approaches, given that efficient engines for disjunctive/unstratified Datalog are available.

While we focus on $\mathcal{SH}$, the method an be extended to richer DLs. Indeed, once we obtain the *knot representation* of a terminology, the algorithm works on the knots and does not depend much on the constructs of the logic. The knot technique thus opens an interesting perspective that might be exploited for other purposes as well.

The rest of this paper is organized as follows. The next section provides basic concepts and notation. After that, we consider in Section 3 forest models on which we can concentrate for our purposes. In Section 4 we introduce knots and discuss how forest-shaped models of a DL knowledge base can be represented using knots. In Section 5, we present our algorithm for answering CQs using knots in the general case, while in Section 6 we address complexity issues and restricted cases. In the final Section 7, we first discuss related work and possible extensions of the approach, and then conclude with some open issues.

2 Preliminaries

In this section we introduce the DL $\mathcal{SH}$ and define the conjunctive query answering problem.

$$\top^{\mathcal{I}} = \Delta^{\mathcal{I}} \quad \bot^{\mathcal{I}} = \emptyset \quad (\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$$
$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}} \qquad (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$$
$$(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y. \langle x,y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$$
$$(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y. \langle x,y\rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$$

Figure 1: Semantics of $\mathcal{SH}$ concepts.

2.1 Description Logic $\mathcal{SH}$

We assume countably infinite sets $\mathbf{C}$, $\mathbf{R}$ and $\mathbf{I}$ of *concept names*, *roles*, and *individuals* respectively. Furthermore, we assume an infinite set $\mathbf{R}^+ \subseteq \mathbf{R}$ of *transitive roles*.

Concepts (in $\mathcal{SH}$) are inductively defined as follows:

(a) $\top$, $\bot$ and every concept name $A \in \mathbf{C}$ is a concept, and

(b) if C, D are concepts and $R \in \mathbf{R}$ is a role, then $C \sqcap D$, $C \sqcup D$, $\neg C$, $\forall R.C$, $\exists R.C$ are concepts.

Let C, D be concepts, R, S be roles and a, b be individuals. Then an expression $C \sqsubseteq D$ is called a *general concept inclusion axiom (GCI)*, an expression $R \sqsubseteq S$ is a *role inclusion axiom (RI)*, while expressions $a : C$ and $\langle a, b\rangle : R$ are *assertions*.

An $\mathcal{SH}$ *knowledge base* (KB) is a pair $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$, where the *TBox* $\mathcal{T}$ is a finite set of GCIs and RIs, while the *ABox* $\mathcal{A}$ is a finite set of assertions. W.l.o.g. we assume that $\mathcal{A} \neq \emptyset$ and that all the concept names and roles occurring in $\mathcal{A}$ also occur in $\mathcal{T}$. By $\mathbf{C}(\mathcal{T})$ and $\mathbf{R}(\mathcal{T})$ we denote the sets of all concept names and roles occurring in a TBox $\mathcal{T}$, respectively. Moreover, we let $\mathbf{R}^+(\mathcal{T}) = \mathbf{R}^+ \cap \mathbf{R}(\mathcal{T})$ and denote by $\sqsubseteq^*_{\mathcal{T}}$ the reflexive transitive closure of $\{(S, R) \mid S \sqsubseteq R \in \mathcal{T}\}$. A role R is *simple* (in a TBox $\mathcal{T}$), if no $S \sqsubseteq^*_{\mathcal{T}} R$ exists such that $S \in \mathbf{R}^+(\mathcal{T})$. Finally, let $\mathbf{I}(\mathcal{A})$ denote the set individuals occurring in an ABox $\mathcal{A}$.

An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ for a KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$ consists of a non-empty *domain* $\Delta^{\mathcal{I}}$ and a *valuation function* $\cdot^{\mathcal{I}}$ that maps each individual $c \in \mathbf{I}(\mathcal{A})$ to an element $c^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, each concept name $C \in \mathbf{C}(\mathcal{T})$ to a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and each role $R \in \mathbf{R}(\mathcal{T})$ to a subset $R^{\mathcal{I}}$ of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The function $\cdot^{\mathcal{I}}$ is extended to all concepts via the equations in Figure 1. We say $\mathcal{I}$ is a *model* of $\mathcal{K}$ (in symbols, $\mathcal{I} \models \mathcal{K}$) if (i) for each GCI $C \sqsubseteq D \in \mathcal{T}$, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$; (ii) for each RI $R \sqsubseteq S \in \mathcal{T}$, $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$; (iii) for each $R \in \mathbf{R}^+(\mathcal{T})$, $R^{\mathcal{I}} = (R^{\mathcal{I}})^+$, i.e., $R^{\mathcal{I}}$ is transitively closed; (iv) for each assertion $a : C$ in $\mathcal{A}$, $a^{\mathcal{I}} \in C^{\mathcal{I}}$; and (v) for each assertion $\langle a, b\rangle : R$ in $\mathcal{A}$, $\langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \in R^{\mathcal{I}}$. If $\mathcal{K}$ admits at least one model, then $\mathcal{K}$ is *satisfiable*.

Example 1. Consider a simple genealogy knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$, where $\mathbf{C}$ contains the concept names man, $woman$, $person$, and $royal$, and $\mathbf{R}$ the roles $father$, $mother$, $parent$, $ancestor$, and $heir$, where $ancestor$ is transitive.

The TBox $\mathcal{T}$ may contain the following GCIs:

$man \sqsubseteq \neg woman,$ which states disjointness of men and women;

$man \sqcup woman \sqsubseteq person, \;\; person \sqsubseteq man \sqcup woman,$ which defines $person$ as men and women; and

$person \sqsubseteq \exists father.man, \;\; person \sqsubseteq \exists mother.woman$ which state that every person has a father and a mother.

The TBox may further contain the RIs

$$father \sqsubseteq parent, \;\; mother \sqsubseteq parent, \;\; parent \sqsubseteq ancestor, \;\; ancestor \sqsubseteq heir.$$

Thus, $father$, $mother$, and $parent$ are simple roles, while $ancestor$ and $heir$ are non-simple.

The ABox $\mathcal{A}$ contains the following assertions:

$$\begin{array}{l} joe : man, \;\; jill : woman, \;\; jill : royal, \\ \langle joe, bob\rangle : father, \;\; \langle bob, sue\rangle : mother, \;\; \langle jill, sue\rangle : mother, \;\; \langle jill, alice\rangle : heir. \end{array}$$

Note that the information about the individuals is incomplete (e.g., joe's father is not known, nor whether he is royal).

A concept C is in *negation normal form (NNF)* if negation occurs in front of the atomic concepts only. As well know, each concept C can be transformed into an equivalent concept in NNF in linear time. We assume that concepts in TBoxes are alway in NNF. As usual, $\sim C$ denotes the NNF of $\neg C$. By $\mathsf{clos}(\mathcal{T})$ we will denote the *concept closure of* $\mathcal{T}$, which is the smallest set such that: (a) $C, D \in \mathsf{clos}(\mathcal{T})$ for each $C \sqsubseteq D \in \mathcal{T}$, (b) if E is a subconcept of $C \in \mathsf{clos}(\mathcal{T})$, then $E \in \mathsf{clos}(\mathcal{T})$, (c) $C \in \mathsf{clos}(\mathcal{T})$, then ${\sim}C \in \mathsf{clos}(\mathcal{T})$, and (d) $\forall R.C \in \mathsf{clos}(\mathcal{T})$ and $R' \in \mathbf{R}(\mathcal{T})$, then $\forall R'.C \in \mathsf{clos}(\mathcal{T})$.

2.2 Conjunctive Query Answering

Let $\mathbf{V}$ be a countably infinite set of variables. A *conjunctive query* (CQ, or *query*) q over a KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$ is a finite set of atoms of the form $A(x)$ or $R(x, y)$, where $A \in \mathbf{C}(\mathcal{T})$, $R \in \mathbf{R}(\mathcal{T})$ and $x, y \in \mathbf{V}$; the set of variables occurring in q is denoted by $\mathbf{V}(q)$. Each CQ q is associated with a unique (possibly empty) tuple $\vec{x} = \langle x_1, \ldots, x_n\rangle$ of *answer variables* from $\mathbf{V}(q)$.

A *match for* q *in an interpretation* $\mathcal{I}$ *for* $\mathcal{K}$ is a mapping $\theta : \mathbf{V}(q) \to \Delta^{\mathcal{I}}$ such that (i) $\theta(x) \in A^{\mathcal{I}}$ for each $A(x) \in q$, and (ii) $\langle \theta(x), \theta(y)\rangle \in R^{\mathcal{I}}$ for each $R(x, y) \in q$. A tuple $\vec{c} = \langle c_1, \ldots, c_n\rangle$ of individuals from $\mathbf{I}(\mathcal{A})$ (of the same arity as $\vec{x}$) *is an answer of* q *over* $\mathcal{I}$, if $\langle c_1^{\mathcal{I}}, \ldots, c_n^{\mathcal{I}}\rangle = \langle \theta(x_1), \ldots, \theta(x_n)\rangle$ for some match θ for q in $\mathcal{I}$; $\mathsf{ans}(q, \mathcal{I})$ denotes the set of all answers of q over $\mathcal{I}$. Then the *answer of* q *over* $\mathcal{K}$ is $\mathsf{ans}(q, \mathcal{K}) = \bigcap_{\mathcal{I} \models \mathcal{K}} \mathsf{ans}(q, \mathcal{I})$, i.e., consists of all tuples $\vec{c}$ that occur in the answer of q for every model $\mathcal{I}$ of $\mathcal{K}$.

Example 2. A possible CQ to the knowledge base $\mathcal{K}$ from Example 1 is

$$q = \{ancestor(x, z), ancestor(y, z), royal(z)\}$$

with answer variables $\vec{x} = \langle x, y\rangle$, which retrieves the individuals that have a common royal ancestor. As easily seen, the query has no answer over $\mathcal{K}$ (i.e., $\mathsf{ans}(q, \mathcal{K}) = \emptyset$), as for each pair $\vec{c} = \langle c_1, c_2\rangle$ of individuals from joe, $jill$, bob, sue, and $alice$, the KB $\mathcal{K}$ has some model $\mathcal{I}$ in which q has no match θ such that $\langle c_1^{\mathcal{I}}, c_2^{\mathcal{I}}\rangle = \langle \theta(c_1), \theta(c_2)\rangle$.

Note that we do not allow for individuals or complex concepts in queries. This is no restriction: if q is a query with individuals, for each individual a we can use a new concept name C_a, replace a in q by a new variable y, and add $C_a(y)$ to q and $a : C_a$ to $\mathcal{A}$. Similarly, atoms $D(x)$, where D is complex, can be simulated by adding a GCI $D \sqsubseteq C_D$ to $\mathcal{K}$ for some fresh concept name C_D, and replacing $D(x)$ by $C_D(x)$.

The query graph of a query q is the directed graph with nodes $\mathbf{V}(q)$ and an arc $x \to y$ for each $R(x, y) \in q$. We say q is *connected* if its query graph is connected,

From now on, we make the *Unique Name Assumption (UNA)*, i.e., in models $\mathcal{I}$ of $\mathcal{K}$, for each pair of individuals $a \neq b$ from $\mathcal{K}$ we have $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. This is not a limitation: as easily seen, UNA does not affect the set of query answers in the case of $\mathcal{SH}$ KBs.

3 Forest Models

It is well know that to answer queries over many known DLs it suffices to restrict the attention to a certain class of models, and, in particular, to *forest-shaped* models. To discuss this, we adopt some notation and naming from [16].

Definition 1 (Trees and Forests)**.** Let $\mathbb{N}^*$ be the set of words over the set $\mathbb{N}$ of natural numbers. We say a set $T \subseteq \mathbb{N}^*$ is a *tree*, if it is prefix closed, i.e., for each word $w{\cdot}e \in T$, where $w \in \mathbb{N}^*$ and $e \in \mathbb{N}$, we have $w \in T$. The empty word ϵ is the root of T, while for each $w \in T$, the nodes $w{\cdot}e \in T$ with $e \in \mathbb{N}$ are *children of* w.

Let $\{T_i\}_{i\in I}$ be a set of trees indexed by I, then the set $F = \bigcup_{i\in I} \big\{(i,w) \mid w \in T_i\big\}$ is called a *forest (with index set I)*. The notion of children is generalized to forests: $(i',w') \in F$ is a *child of* $(i,w) \in F$ if $i = i'$ and w' is a child of w. Similarly, each node (i,ϵ) is a root of F.

An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ for a KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$ is *forest-shaped*, if

(a) the domain $\Delta^{\mathcal{I}}$ is a forest with index set $\mathbf{I}(\mathcal{A})$,

(b) for each $d, e \in \Delta^{\mathcal{I}}$ and role R such that $(d,e) \in R^{\mathcal{I}}$, either e is a child of d, or both d and e are root nodes, and

(c) for each node $d \in \Delta^{\mathcal{I}}$, the number of children of d is bounded by $|\mathsf{clos}(T)|$.

By $\mathsf{children}(\mathcal{I}, e)$ we denote the set of children of $e \in \Delta^{\mathcal{I}}$. We say $\mathcal{I}$ is *tree-shaped* if $|\mathbf{I}(\mathcal{A})| = 1$. For a tree-shaped $\mathcal{I}$, let $\mathsf{root}(\mathcal{I})$ denote the unique root node of $\mathcal{I}$.

To ease presentation, as tree-shaped we also consider any interpretation that is isomorphic to a tree-shaped interpretation defined above; the two functions $\mathsf{children}(\cdot,\cdot)$ and $\mathsf{root}(\cdot)$ are extended accordingly. We will further use $\mathcal{I}_{|e}$ to denote the tree-shaped interpretation obtained by restricting a forest-shaped $\mathcal{I}$ to $e \in \Delta^{\mathcal{I}}$ and its descendants, i.e., $\mathcal{I}_{|e}$ is the subtree of $\mathcal{I}$ rooted at e.

In presence of transitivity statements, we strictly speaking do not have forest-shaped models of an $\mathcal{SH}$ KB in general. Indeed, by definition, forest-shaped models cannot have transitive arcs. In order to provide a complete query answering algorithm, we impose additional constraints on forest-shaped models.

Definition 2 (Forest base and closure)**.** A *forest-base* for a KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$ is any forest-shaped interpretation $\mathcal{I}$ for $\mathcal{K}$ such that:

(a) for each GCI $C \sqsubseteq D \in \mathcal{T}$, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$;

(b) for each RI $R \sqsubseteq S \in \mathcal{T}$, $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$;

(c) for each assertion $a : A$ (resp., $\langle a,b\rangle : R$) in $\mathcal{A}$, we have $a^{\mathcal{I}} \in A^{\mathcal{I}}$ (resp., $\langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \in R^{\mathcal{I}}$);

(d) if $e \in (\forall R.C)^{\mathcal{I}}$, then for all $S \in \mathbf{R}^+(\mathcal{T})$ with $S \sqsubseteq^*_{\mathcal{T}} R$ we also have $e \in (\forall S.(\forall S.C))^{\mathcal{I}}$.

The *closure* of $\mathcal{I}$ is the interpretation $\mathcal{J} = \langle \Delta^{\mathcal{J}}, \cdot^{\mathcal{J}}\rangle$ that is identical to $\mathcal{I}$ except that, for each role R,

$$R^{\mathcal{J}} = R^{\mathcal{I}} \cup \bigcup_{S \sqsubseteq^*_{\mathcal{T}} R \wedge S \in \mathbf{R}^+(\mathcal{T})} (S^{\mathcal{I}})^+.$$

Forest-bases of $\mathcal{K}$ satisfy all the axioms and assertions of $\mathcal{K}$ (conditions (a-c)). However, they are not necessarily models of $\mathcal{K}$ since the transitivity requirements may be violated. To deal with this, we require (d) which emulates the effect of transitive roles on a model. The closure, which is obtained by closing a forest-base under transitivity and role hierarchy, leads us to a model of a KB.

We can now state the following important proposition.

Proposition 1 ([16])**.** *If $\mathcal{J}$ is the closure of a forest-base $\mathcal{I}$ for a KB $\mathcal{K}$, then $\mathcal{J}$ is a model of $\mathcal{K}$. Moreover, given a KB $\mathcal{K}$, a query q and a tuple $\vec{c}$ of individuals, if $\vec{c} \notin \mathsf{ans}(q, \mathcal{K})$, then there exists some forest-base $\mathcal{I}$ for $\mathcal{K}$ such that $\vec{c} \notin \mathsf{ans}(q, \mathcal{J})$, where $\mathcal{J}$ is the closure of $\mathcal{I}$.*

The proposition above implies that we can safely concentrate on closures of forest-bases for answering CQs. In fact, we will look for query mappings in forest bases instead of their closures. This will not be a limitation, as we just need a slightly relaxed version of matches.

Definition 3 (Prematches)**.** Given a forest interpretation $\mathcal{I}$ and $d_1, d_n \in \Delta^{\mathcal{I}}$, we call d_n an R-successor of d_1 (in $\mathcal{I}$), if there is some $S \sqsubseteq^* R$ and a sequence $d_1, \ldots, d_n$ such that $(d_i, d_{i+1}) \in S^{\mathcal{I}}$ for each $1 \leq i < n$, and $n > 2$ implies that S is transitive. We say a query q has a *pre-match in $\mathcal{I}$*, if there is a mapping $\pi : \mathbf{V}(q) \rightarrow \Delta^{\mathcal{I}}$ such that:

(PM1) $A(x) \in q$ implies $\pi(x) \in A^{\mathcal{I}}$, and

(PM2) $R(x, y) \in q$ implies $\pi(y)$ is an R-successor of $\pi(x)$ in $\mathcal{I}$.

The following is then a direct consequence of Proposition 1 and Definition 3 above.

Proposition 2. *Given a KB $\mathcal{K}$, a query q with answer variables $\vec{x} = \langle x_1, \ldots, x_n \rangle$, and a tuple $\vec{c} = \langle c_1, \ldots, c_n \rangle$ of individuals, it holds that $\vec{c} \in \mathsf{ans}(q, \mathcal{K})$ iff in each forest-base $\mathcal{I}$ for $\mathcal{K}$ there exists a pre-match π for q such that $\langle c_1^{\mathcal{I}}, \ldots, c_n^{\mathcal{I}} \rangle = \langle \pi(x_1), \ldots, \pi(x_n) \rangle$.*

By the above proposition, to answer a query it suffices to look at prematches in forest-bases only.

4 Model Representation via Knots

We deal here with model representation, and provide a method to finitely represent the possibly infinite forest-bases of an $\mathcal{SH}$ KB. This will be the basis of our query answering algorithm employing knowledge compilation.

Before dealing with full forest-bases, we first we present *knots*, which are special labeled trees of depth ≤ 1 used to represent the tree parts of forest-bases. For the rest of this section, we assume a fixed $\mathcal{SH}$ terminology $\mathcal{T}$, and all ABoxes that we consider are ABoxes for $\mathcal{T}$ (i.e., over the signature of $\mathcal{T}$).

Definition 4 (Knots)**.** A *type* (for $\mathcal{T}$) is any set $\tau \subseteq \mathsf{clos}(\mathcal{T})$. A *knot* (for $\mathcal{T}$) is any tuple (r, S), where $r \subseteq \mathsf{clos}(\mathcal{T})$ and $S \subseteq 2^{\mathbf{R}(\mathcal{T})} \times \mathsf{clos}(\mathcal{T})$, satisfying the following consistency conditions:

(a) $\sim C \sqcup D \in r$ for each GCI $C \sqsubseteq D \in \mathcal{T}$;

(b) if $C \in r$, then $\sim C \notin r$;

(c) if $C \sqcap D \in r$, then $C, D \in r$;

(d) if $C \sqcup D \in r$, then $C \in r$ or $D \in r$;

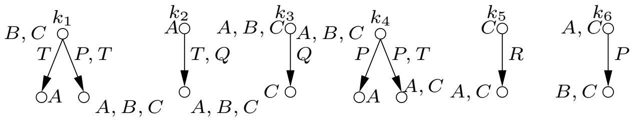

Figure 2: Example set of knots

(e) if $\forall R.C \in r$, then for each $(\alpha, \beta) \in S$ with $R \in \alpha$, we have $C \in \beta$;

(f) if $\forall R.C \in r$, then for each $R' \sqsubseteq^* R$ with $R' \in \mathbf{R}^+(\mathcal{T})$ and each $(\alpha, \beta) \in S$ with $R' \in \alpha$, we have $\forall R'.C \in \beta$;

(g) if $\exists R.C \in r$, then there exists $(\alpha, \beta) \in S$ with $R \in \alpha$ and $C \in \beta$;

(h) if $R \sqsubseteq R'$ is a role inclusion in $\mathcal{T}$, then for each $(\alpha, \beta) \in S$ with $R \in \alpha$, we have $R' \in \alpha$;

(i) $|S| \leq \mathsf{clos}(\mathcal{T})$.

We say r is the *root* and the elements in S are the *children* in the knot (r, S).

Knots are self-contained model building blocks for forest-bases of $\mathcal{K}$: a knot (r, S) can be viewed as an abstract element of a forest-base for $\mathcal{T}$ that satisfies the concepts in r, and for each $(\alpha, \beta) \in S$ has a successor linked by roles in α and satisfying concepts in β. Such a knot (r, S) encodes a possible combination of immediate successors for a node having type r in a forest-base. Note that $|\mathsf{clos}(\mathcal{T})|$ is polynomial in the size of $\mathcal{T}$, and hence we can construct at most exponentially many different knots for $\mathcal{T}$ (we discuss this in Section 6).

We need some global conditions on *knot sets* to ensure that trees can be built out of knots.

Definition 5 (Consistency)**.** A set K of knots for $\mathcal{T}$ is *consistent*, if for each $(r, S) \in K$ and each $(\alpha, \beta) \in S$ there exists some $(r', S') \in K$ such that $\beta = r'$. Such a knot (r', S') is called a *possible successor* of (r, S) (in K).

Example 3. A consistent set of knots (for some KB $\mathcal{K}$) is depicted in Figure 2. Graphically, we represent a knot $k = (r, S)$ as a tree where the root is labeled r, and that has an arc labeled α to a child labeled β for each $(\alpha, \beta) \in S$; for simplicity, parentheses "$\{$" and "$\}$" are omitted. Five different types $\tau_X = \{X\} \subseteq \{A, B, C\}$ occur in these knots, viz. τ_A, τ_C, $\tau_{A,C}$, $\tau_{B,C}$, and $\tau_{A,B,C}$. Observe that k_2 is a possible successor of $(\{T\}, \tau_A)$ in k_1, while $(\{P, T\}, \tau_{A,B,C})$ has the possible successors k_3 and k_4.

We can build trees by putting suitable knots together subsequently. Intuitively, consistency of a knot set means that for each knot we have a possible successor knot, and hence the tree construction will not fail. In order to deal with ABoxes, we will need knot sets that can be used to build trees starting at the ABox individuals.

Definition 6 (Compatibility)**.** Let Q be a set of types for $\mathcal{T}$. Then a consistent set K of knots for $\mathcal{T}$ is *Q-compatible*, if for each $\tau \in Q$ there exists some $(r, S) \in K$ with $r = \tau$.

Intuitively, if K is Q-compatible, then for each $\tau \in Q$ there exists a knot in K which can be used as a "starting" knot for constructing a tree with root τ. We can now turn to ABoxes.

Definition 7 (ABox completions). An ABox $\mathcal{A}'$ is called a *completion* of an ABox $\mathcal{A}$ for $\mathcal{T}$, if the following holds:

(a) $\mathcal{A} \subseteq \mathcal{A}'$;

(b) $a : {\sim}C \sqcup D \in \mathcal{A}'$ for each concept inclusion $C \sqsubseteq D \in \mathcal{T}$ and individual a of $\mathcal{K}$;

(c) if $a : C \in \mathcal{A}'$, then $a : {\sim}C \notin \mathcal{A}'$;

(d) if $a : C \sqcap D \in \mathcal{A}'$ (resp., $a : C \sqcup D \in \mathcal{A}'$), then $a : C \in \mathcal{A}'$ and $a : D \in \mathcal{A}'$ (resp., $a : C \in \mathcal{A}'$ or $a : D \in \mathcal{A}'$);

(e) if $a : \forall R.C \in \mathcal{A}'$ and $(a, b) : R \in \mathcal{A}'$, then $b : C \in \mathcal{A}'$;

(f) if $R \sqsubseteq S$ is a role inclusion in $\mathcal{T}$ and $(a, b) : R \in \mathcal{A}'$, then $(a, b) : S \in \mathcal{A}'$;

(g) if $(a, b) : R \in \mathcal{A}'$, $(b, c) : R \in \mathcal{A}'$ and $R \in \mathbf{R}^+(\mathcal{T})$, then $(a, c) : R \in \mathcal{A}'$.

The set of completions of $\mathcal{A}$ is denoted by $\mathsf{comp}(\mathcal{A})$. For any ABox $\mathcal{A}$ let $\mathcal{A}(a) = \{C \mid a : C \in \mathcal{A}\}$ and $C_{\mathcal{A}} = \{\mathcal{A}(a) \mid a \in \mathbf{I}(\mathcal{A})\}$, i.e., the type of a in $\mathcal{A}$ and the set of types of individuals in $\mathcal{A}$, respectively. Finally, we define $C^{\mathcal{T}}_{\mathcal{A}} = \bigcup_{\mathcal{A}' \in \mathsf{comp}(\mathcal{A})} C_{\mathcal{A}'}$, which is the set of types occurring in completions of $\mathcal{A}$.

An ABox completion corresponds to a possible explication of the constraints on its individuals given by the terminology, where the existential restrictions are dispensed. The ABox completions provide us with the graph parts of forest-bases, which can be characterized in terms of ABox completions and compatible knot sets. We first give a construction of forest bases.

Definition 8 (Induced forest interpretations). Let $\mathcal{A}$ be an ABox and let K be a $C_{\mathcal{A}}$-compatible knot set. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is *induced by* $\mathcal{A}$ and K, if $\Delta^{\mathcal{I}}$ is a forest with index set $\mathbf{I}(\mathcal{A})$ and $\cdot^{\mathcal{I}}$ is such that:

(a) For each $(a_1, \epsilon), (a_2, \epsilon) \in \Delta^{\mathcal{I}}$ and role R, we have $((a_1, \epsilon), (a_2, \epsilon)) \in R^{\mathcal{I}}$ iff $\langle a_1, a_2 \rangle : R \in \mathcal{A}$.

(b) There exists a mapping $\varphi : \Delta^{\mathcal{I}} \to K$ such that for each element $e \in \Delta^{\mathcal{I}}$ the knot $\varphi(e) = (r, S)$ satisfies:

- for each atomic concept A, $e \in A^{\mathcal{I}}$ iff $A \in r$, and
- there exists a bijection $f : S \to \mathsf{children}(\mathcal{I}, e)$ such that for each $s = (\alpha, \beta)$ in S and each role R, we have $(e, f(s)) \in R^{\mathcal{I}}$ iff $R \in \alpha$.

The set of all such $\mathcal{I}$ is denoted $\mathfrak{F}(\mathcal{A}, K)$.

The following is a direct consequence of the above definitions.

Proposition 3. *If $\mathcal{A}'$ is a completion of an ABox $\mathcal{A}$ and K is $C_{\mathcal{A}'}$-compatible, then each $\mathcal{I} \in \mathfrak{F}(\mathcal{A}', K)$ is a forest-base for the KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$.*

We saw that we can generate *some* forest-bases. In fact, we want to capture all forest-bases of a given terminology, and for this we introduce the notion of completeness.

Definition 9 (Completeness). For a knot set K and type τ, let $K|\tau$ denote the smallest subset of K such that

(a) $(r, S) \in K|\tau$ for each $(r, S) \in K$ with $r = \tau$, and

(b) if $(r, S) \in K|\tau$, $(\alpha, \beta) \in S$ and $(r', S') \in K$ is such that $\beta = r'$, then $(r', S') \in K|\tau$, i.e., $K|\tau$ is closed under the possible successors in K.

Let K be a consistent set of knots for $\mathcal{T}$ and Q a set of types for $\mathcal{T}$. We say K is *Q-complete* if for each $\tau \in Q$ and each consistent set K' of knots for $\mathcal{T}$ we have $K'|\tau \subseteq K$.

Intuitively, $K|\tau$ is the restriction of K to knots that have root τ, or are reachable from the former via the possible successor relation. A Q-complete knot set contains all knots that can be used to build a tree starting with a type $\tau \in Q$.

Definition 10 (Induced forest-bases)**.** Let $\mathcal{A}$ be an ABox and let K be a $C^{\mathcal{T}}_{\mathcal{A}}$-complete knot set. Then we denote by $\mathfrak{F}_K(\mathcal{A})$ the set of forest-bases induced by a completion of $\mathcal{A}$ and the knot set K.

It remains to see that in order to answer a query over a KB, it suffices to look at the induced forest-bases.

Proposition 4. *Suppose $\mathcal{A}$ is an ABox, K a $C^{\mathcal{T}}_{\mathcal{A}}$-complete knot set, q a query with answer variables $\vec{x} = \langle x_1, \dots, x_n \rangle$, and $\vec{c} = \langle c_1, \dots, c_n \rangle$ a tuple of individuals. Then $\vec{c} \in \mathsf{ans}(q, (\mathcal{T}, \mathcal{A}))$ iff in each forest-base $\mathcal{I} \in \mathfrak{F}_K(\mathcal{A})$ there exists a prematch π for q such that $\langle c_1^{\mathcal{I}}, \dots, c_n^{\mathcal{I}} \rangle = \langle \pi(x_1), \dots, \pi(x_n) \rangle$.*

Proof. The "$\rightarrow$" direction follows directly from Proposition 2.

For the "$\leftarrow$" direction, assume $\vec{c} \notin \mathsf{ans}(q, (\mathcal{T}, \mathcal{A}))$. By Proposition 2, there exists a forest-base $\mathcal{J}$ for $(\mathcal{T}, \mathcal{A})$ that admits no prematch π for q with $\langle c_1^{\mathcal{J}}, \dots, c_n^{\mathcal{J}} \rangle = \langle \pi(x_1), \dots, \pi(x_n) \rangle$. We just need to argue that $\mathcal{J} \in \mathfrak{F}_K(\mathcal{A})$. To this end, we decompose $\mathcal{J}$ into a completion of $\mathcal{A}$ and a set of knots.

Let $\mathcal{A}'$ be the smallest ABox such that: (a) $\mathcal{A} \subseteq \mathcal{A}'$, (b) if $C \in \mathsf{clos}(\mathcal{T})$ and $a^{\mathcal{J}} \in C^{\mathcal{J}}$, then $a : C \in \mathcal{A}'$, (c) if $(a^{\mathcal{J}}, b^{\mathcal{J}}) \in R^{\mathcal{J}}$, then $\langle a, b \rangle : R \in \mathcal{A}'$, (d) if $(a, b) : R \in \mathcal{A}'$, $(b, c) : R \in \mathcal{A}'$ and $R \in \mathbf{R}^+(\mathcal{T})$, then $(a, c) : R \in \mathcal{A}'$, and (e) if $R \sqsubseteq S$ is a role inclusion in $\mathcal{T}$ and $(a, b) : R \in \mathcal{A}'$, then $(a, b) : S \in \mathcal{A}'$. It is easy to see that since $\mathcal{J}$ is a forest-base, $\mathcal{A}'$ is a completion of $\mathcal{A}$.

We now "decompose" the tree parts of $\mathcal{J}$ into knots. Let φ be a mapping that assigns to each $e \in \Delta^{\mathcal{J}}$ a knot $\varphi(e) = (r, S)$, where

(a) $r = \{C \in \mathsf{clos}(\mathcal{T}) \mid e \in C^{\mathcal{J}}\}$, and

(b) $S = \bigcup_{i \in I} \{(\alpha_i, \beta_i)\}$, where $I = \{e{\cdot}x \in \Delta^{\mathcal{I}} \mid x \in \mathbb{N}\}$, $\alpha_i = \{R \mid (e, i) \in R^{\mathcal{J}}\}$ and $\beta_i = \{C \in \mathsf{clos}(\mathcal{T}) \mid i \in C^{\mathcal{J}}\}$.

By the above mapping we assign to each $e \in \Delta^{\mathcal{J}}$ a knot extracted from $\mathcal{J}$ itself. By construction, the knot set $K' = \{\varphi(e) \mid e \in \Delta^{\mathcal{J}}\}$ is consistent. Furthermore, K' is $C_{\mathcal{A}'}$-compatible and $K' \subseteq K$ holds due to $C^{\mathcal{T}}_{\mathcal{A}}$-completeness of K. Finally, the mapping φ witnesses (see Definition 8) that $\mathcal{J}$ is a forest-base induced by $\mathcal{A}'$ and K, i.e., $\mathcal{J} \in \mathfrak{F}_K(\mathcal{A})$. $\square$

We finally note that given a given KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ and a set Q of types for $\mathcal{T}$, we can easily compute a Q-complete knot set. We can do this via a simple procedure inspired by type-elimination [32]. We start by computing the set K of all knots for $\mathcal{T}$ that have root $\tau \in Q$. In the second stage, we close K under the possible successor knots. Finally, we remove from K one by one the knots that have a leaf for which K does not provide at least one successor knot. We elaborate on this with the algorithm presented in Figure 3. As easily seen, the algorithm returns a desired knot set. Indeed, for any type $\tau \in Q$ and a consistent knot set K, the algorithm will include $K|\tau$, and none of the knots from $K|\tau$ will be deleted in the second stage which is designed to ensure consistency. We note that a Q-complete knot set can be obtained in single exponential time in the size of $\mathcal{T}$ (we elaborate on this in Section 6).

Algorithm 1: computeKnots

```
Data: a KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, a set $Q$ of types for $\mathcal{T}$
Result: a $Q$-complete knot set $K$
begin
    Build the set $K$ of all knots $(r, S)$ for $\mathcal{T}$ such that $r \in Q$;
    Close $K$ under the following rule:
        if $(r, S) \in K$ and $(\alpha, \beta) \in S$, then add to $K$ each knot $(\tau', S')$ for $\mathcal{T}$ such that $\tau' = \beta$.
    repeat
        Let $K' := K$;
        if $(r, S) \in K$ and $(\alpha, \beta) \in S$ but there exists no knot $(\tau', S') \in K$ s.t. $\tau' = \beta$ then
            $K := K \setminus \{(r, S)\}$;
    until $K' \neq K$ ;
    return $K$
end
```

Figure 3: Building knot sets.

5 Query Answering with Knots

We now present our algorithm for answering conjunctive queries over $\mathcal{SH}$ knowledge bases. The method relies on knot sets and is presented in three steps:

- We first consider the structure of a *query prematch* in a forest-shaped interpretation, and based on this structure define a notion of *subqueries* and their matches.
- We then compile an input query q and a terminology $\mathcal{T}$ into a *type-query table*, which, informally speaking, tells which subqueries of q can be mapped in any tree generated from knots starting with a particular root type.
- Finally, given an arbitrary ABox $\mathcal{A}$, we can answer q over a KB $\langle \mathcal{T}, \mathcal{A} \rangle$ by considering partial mappings of q into completions of $\mathcal{A}$ and by looking up the query remainders in the precomputed type-query table.

To ease presentation, for the rest of this section we fix an $\mathcal{SH}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ and a CQ q. Let us also assume any $C^{\mathcal{T}}_{\mathcal{A}}$-complete knot set K, and let $\mathsf{types}(K) = \{r \mid (r, S) \in K\}$.

5.1 Subqueries and Rooted Matches

We define a notion of subqueries and their matches in tree-shaped interpretations, which will allow us to construct a full prematch for a query in a forest-base out of matches for subqueries in trees and a partial query mapping into the graph part of the forest-base.

Suppose $\mathcal{I}$ is a forest-base for $\mathcal{K}$ in which a query q has a prematch π, and $e \in \Delta^{\mathcal{I}}$ is arbitrary but not a root node in $\mathcal{I}$. Let $V_e \subseteq \mathbf{V}(q)$ be the set of variables of q that are mapped at e or inside the subtree of $\mathcal{I}$ rooted at e. We can make the following observations about the variables of q that must be mapped in the latter subtree:

(P1) If $x \in V_e$ and q contains an atom $R(x, y)$, then $y \in V_e$, i.e., y must be mapped into the subtree of $\mathcal{I}$ rooted at e.

(P2) If q contains two atoms $R(x, y)$ and $R'(x', y)$ where R and R' are simple (in $\mathcal{K}$), and $x, y \in V_e$, then $x' \in V_e$, i.e., x must also be mapped into the same subtree, and in particular at the same element as x.

(P3) If $y \in V_e$ and there is $R(x, y) \in q$ with simple R and such that $x \notin V_e$, then y must be mapped to e.

(P4) If $y \in V_e$ and there is $R(x, y) \in q$ where R is not simple, and $x \notin V_e$, then $\pi(y) = e$ or $\pi(y)$ is an R-successor of e.

The above observations are reflected in the notions of subqueries and their matches in tree-shaped interpretations.

Definition 11 (Subqueries)**.** Given $X \subseteq \mathbf{V}(q)$ and $y \in X$, let $\mathsf{back}(X, y) = \{R \mid R(x, y) \in q \wedge x \notin X\}$. We call $y \in X$ *open* in X if $\mathsf{back}(X, y) \neq \emptyset$. If in addition $\mathsf{back}(X, y)$ does not contain a simple role, then y is *free* in X.

An *f(orward)-subquery* of q is any a tuple (X, Σ) where

(a) $X \subseteq \mathbf{V}(q)$ is a set of variables obeying the following rules:

  (i) If $R(x, y) \in q$ and $x \in X$, then $y \in X$.

  (ii) if $R(x, y) \in q$ and $R'(x', y) \in q$ are two atoms where R, R' are simple, and $x \in X$, then we also have $x' \in X$.

  (iii) The restriction of the query graph of q to variables in X is connected and acyclic, i.e., X induces a connected acyclic subquery of q.

(b) Σ is a mapping that assigns to every free variable y in X some set $\Sigma(y) \subseteq \mathbf{R}^+(\mathcal{K})$ containing some role $T \sqsubseteq^* R$ for each $R \in \mathsf{back}(X, y)$.

The set of f-subqueries of q is denoted by $\mathbb{F}$. Any set $\rho \in 2^{\mathbb{F}}$ of f-subqueries is called a *disjunctive f-subquery* of q.

Example 4. We assume a knowledge base $\mathcal{K}$ in which $\mathbf{R}^+(\mathcal{K}) = \{T\}$, and where $\mathcal{K}$ contains a role inclusion axiom $T \sqsubseteq R$. For our examples, we consider the query

$$q = C(x_1), T(x_1, x_2), Q(x_2, x_3), R(x_3, x_4), A(x_4), C(x_4), P(x_1, x_6), B(x_5), R(x_5, x_6), Q(x_6, x_7), C(x_7),$$

whose query graph, augmented with node labels $\{C \in \mathbf{C} \mid C(x) \in q\}$ and edge labels $\{R \in \mathbf{R} \mid R(x, y) \in Q\}$, is depicted in Figure 4. A largest f-subquery of q is $F = (\mathbf{V}(q), \emptyset)$ (i.e., the full q). In any f-subquery that contains all variables except x_5, the variable x_6 is open (the non-simple R is the only role in $\mathsf{back}(\mathbf{V}(q) \setminus \{x_5\}, x_6)$), and therefore $T \in \Sigma(x_6)$ must hold; $F_1 = (\mathbf{V}(q) \setminus \{x_5\}, \{(x_6, \{T\})\})$ is such an f-subquery. Other f-subqueries with some free variable are $F_2 = (\{x_2, x_3, x_5\}, \{(x_2, \{T\})\})$, $F_3 = (\{x_6, x_7\}, \{(x_6, \{T\})\})$ and $F_4 = (\{x_4\}, \{(x_4, \{T\})\})$. These f-subqueries are also graphically represented in Figure 4, where the Σ-components are omitted.

In an f-subquery (X, Σ), the set X comprises variables that have to be mapped into a given tree-shaped interpretation. The set Σ is designed to deal with the situation (P4) and stores the roles via which free variables of X must be reached from the root of the interpretation. Due to the observations (P1-P2), we can safely require X to be closed under the rules (a.i) and (a.ii). Open variables that are not free must be mapped at the root of the tree due to (P3).

The formal definition of matches for subqueries is as follows:

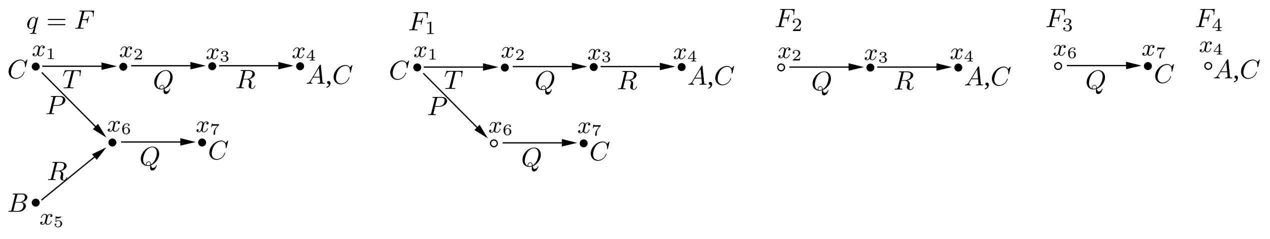

Figure 4: An example query and some of its f-subqueries

Definition 12 (Rooted prematches)**.** Given $(X, \Sigma) \in \mathbb{F}$ and a tree-shaped interpretation $\mathcal{I}$, we write $\mathcal{I} \models (X, \Sigma)$ if there exists a mapping $\pi : X \longrightarrow \Delta^{\mathcal{I}}$ (a *rooted prematch for* (X, Σ) *in* $\mathcal{I}$) that obeys the following rules:

(RP1) If $x \in X$ and $A(x) \in q$, then $\pi(x) \in A^{\mathcal{I}}$;

(RP2) If $x, y \in X$ and $R(x, y) \in q$, then $\pi(y)$ is an R-successor of $\pi(x)$ in $\mathcal{I}$;

(RP3) If y is open in X but it is not free, then $\pi(y) = \mathsf{root}(\mathcal{I})$;

(RP4) If y is a free variable in X and $R \in \Sigma(y)$, then $\pi(y)$ is either the root of $\mathcal{I}$ or an R-successor of the root of $\mathcal{I}$.

Additionally, we write $\mathcal{I} \models^d (X, \Sigma)$ if π is such that for every $y \in X$, the depth of $\pi(y)$ in $\mathcal{I}$ is $\leq d$, i.e., the match is within depth d in $\mathcal{I}$. Furthermore, given a disjunctive f-subquery $\rho \in 2^{\mathbb{F}}$, we write $\mathcal{I} \models \rho$ (resp., $\mathcal{I} \models^d \rho$) if for some $(X, \Sigma) \in \rho$ we have $\mathcal{I} \models (X, \Sigma)$ (resp., $\mathcal{I} \models^d (X, \Sigma)$).

5.2 Subquery Entailment at Knots and Types

In the following, we provide a method to test existence of rooted matches in tree-shaped interpretations constructed out of knots starting with a particular knot or type. The formal definitions of such trees and the entailment problem are as follows.

Definition 13 (k-trees and τ-trees)**.** Let $k \in K$ be a knot and $\tau \in \mathsf{types}(K)$ a type. A tree-shaped interpretation $\mathcal{I}$ is a called a *k-tree* (resp., *τ-tree*), if there exists a mapping $\varphi^{\mathcal{I}} : \Delta^{\mathcal{I}} \rightarrow K$ such that for each $e \in \Delta^{\mathcal{I}}$ the knot $\varphi^{\mathcal{I}}(e) = (r, S)$ satisfies:

(a) if e is the root of $\mathcal{I}$, then $(r, S) = k$ (resp., $r = \tau$),

(b) for each atomic concept A, $e \in A^{\mathcal{I}}$ iff $A \in r$, and

(c) there exists a bijection $b : S \longrightarrow \mathsf{children}(\mathcal{I}, e)$ such that for each $s = (\alpha, \beta)$ in S and each role R, $(e, b(s)) \in R^{\mathcal{I}}$ iff $R \in \alpha$.

The sets of all k-trees and τ-trees are denoted by $\mathfrak{T}(k)$ and by $\mathfrak{T}(\tau)$, respectively.

Note that for every $\mathcal{I} \in \mathfrak{T}(k)$, at the root of $\mathcal{I}$ we have a unique bijection b in (c), and thus each node e at depth 1 (i.e., each child of the root) is uniquely identified by some leaf $s \in S$ of $k = (\tau, S)$; for convenience, we will refer to e by $b^{\mathcal{I}}_s$. We will also use $\mathcal{I}_s$ to denote $\mathcal{I}_{|b^{\mathcal{I}}_s}$.

Definition 14 (Entailment at knots and types)**.** Given $\rho \in 2^{\mathbb{F}}$ and $k \in K$, we write $k \models \rho$ (resp., $k \models^d \rho$) if for each $\mathcal{I} \in \mathfrak{T}(k)$ we have $\mathcal{I} \models \rho$ (resp., $\mathcal{I} \models^d \rho$). Similarly, given $\rho \in 2^{\mathbb{F}}$ and a type $\tau \in \mathsf{types}(K)$, we write $\tau \models \rho$ (resp., $\tau \models^d \rho$) if $\mathcal{I} \models \rho$ (resp., $\mathcal{I} \models^d \rho$) for each $\mathcal{I} \in \mathfrak{T}(\tau)$.

We will use *type-query* and *knot-query* tables to store relevant pairs of types and disjunctive f-subqueries, and pairs of knots and disjunctive f-subqueries for which the entailment relation above holds.

Definition 15 (kq-table and tq-table)**.** a A *knot-query table (kq-table)* is an arbitrary relation $R \subseteq K \times 2^{\mathbb{F}}$. We say R is *(d-)complete,* if (i) $(k, \rho) \in R$ implies $k \models^{(d)} \rho$, and (ii) if $k \models^{(d)} \rho$ and there is no $\rho' \subset \rho$ with $k \models^{(d)} \rho'$, then $(k, \rho) \in R$.

Similarly, a *type-query table (tq-table)* is any relation $R' \subseteq \mathsf{types}(K) \times 2^{\mathbb{F}}$, and we say R' is *(d-)complete* if (i) $(\tau, \rho) \in R'$ implies $\tau \models^{(d)} \rho$, and (ii) if $\tau \models^{(d)} \rho$ and there is no $\rho' \subset \rho$ with $\tau \models^{(d)} \rho'$, then $(\tau, \rho) \in R'$.

In the remainder of this section, we show how to compute a d-complete kq-table, a d-complete tq-table, and, finally, a complete tq-table that will be used to answer queries over the full knowledge base $\mathcal{K}$. The basic strategy is as follows:

(I) we show how to compute a d-complete tq-table TQ^d from a given d-complete kq-table KQ^d, and

(II) we provide a way to obtain, given a d-complete tq-table TQ^d, a $d{+}1$-complete kq-table KQ^{d+1}.

Note that we can easily build a 0-complete tq-table TQ^0, by looking at types $\tau \in \mathsf{types}(K)$: for every variable x in q that has no successors in q and such that $\{A \mid A(x) \in q\} \subseteq \tau$, take all subqueries (X, Σ) where $X = \{x\}$ and, if x is free in X, Σ assigns x some possible set as Definition 11.b (in fact, it is sufficient to take only the single maximal such possible set). Hence, by iteratively applying the two steps above, we can compute d-complete tq-tables and kq-tables for any $d \in \mathbb{N}$. It will be easy to see that in this way we can obtain a compete tq-table.

Example 5. The following table TQ^0 is an example of a 0-complete tq-table for the query q given in Example 4 and the set of knots given in Example 3, where as above $F_4 = (\{x_4\}, \{(x_4, \{T\})\})$ and $F_5 = (\{x_7\}, \emptyset)$. Note that in this case, it is enough to consider singleton disjunctive f-subqueries.

Type	disjunctive f-subquery
τ_C	$\{F_5\}$
$\tau_{A,C}$	$\{F_5\}$
$\tau_{A,C}$	$\{F_4\}$
$\tau_{B,C}$	$\{F_5\}$
$\tau_{A,B,C}$	$\{F_5\}$
$\tau_{A,B,C}$	$\{F_4\}$

The central notion for the computation is that of minimal hitting sets.

Definition 16 (Minimal hitting sets and k/τ-hits)**.** Assume $k \in K$ and $\tau \in \mathsf{types}(K)$. Then a set $h \subseteq \mathbb{F}$ is called a k-*hit* (resp., τ-*hit*) of a kq-table (resp., tq-table) R, if h is a minimal (w.r.t. inclusion) set such that $h \cap \rho \neq \emptyset$ for each $(k', \rho) \in R$ with $k' = k$ (resp., for each $(\tau', \rho) \in R$ with $\tau' = \tau$).

The following property of minimal hitting sets is important.

Algorithm 2: TQ_from_KQ

```
Data: a $d$-complete kq-table KQ
Result: a $d$-complete tq-table TQ
begin
    $R := \emptyset$;
    forall $\tau \in \mathsf{types}(K)$ do
        Compute the set $H = \big\{h \subseteq \mathbb{F} \mid k \in K$ has root $\tau$ and $h$ is a $k$-hit of $\mathsf{KQ}\big\}$;
        forall $\rho \in 2^{\mathbb{F}}$ do
            if for each $h \in H$ we have $h \cap \rho \neq \emptyset$ then
                $R := R \cup \{(\tau, \rho)\}$;
    return $R$
end
```

Figure 5: From knot-query tables to type-query tables (for the knot set K and the query q).

Lemma 1. *Suppose h is a k-hit (resp., a τ-hit) of a d-complete kq-table* KQ *(resp., of a d-complete tq-table* TQ*). Then there exists some $\mathcal{I} \in \mathfrak{T}(k)$ (resp., $\mathcal{I} \in \mathfrak{T}(\tau)$) such that $\mathcal{I} \models^d (X, \Sigma)$ iff $(X, \Sigma) \in h$.*

Proof. We only consider the case of k-hits, the proof for τ-hits is analogous.

Consider a k-hit h as above and consider $\rho = \mathbb{F} \setminus h$. Since $\rho \cap H = \emptyset$, we have $(k, \rho) \notin \mathsf{KQ}$. Hence, $k \not\models^d \rho$, i.e., there is a $\mathcal{I} \in \mathfrak{T}(k)$ such that $\mathcal{I} \not\models^d F$ for each $F \in \rho$. It remains to show that $\mathcal{I}$ is also such that $\mathcal{I} \models^d F$ for all $F \in h$.

Consider an arbitrary $F \in H$. As easily seen, by minimality of h there exists some $(k, \rho_F) \in S$ such that $F \in \rho_F$ and $|\rho_F \cap h| = 1$ (if not, $h \setminus \{F\}$ would be a smaller hitting set). We know that $\mathcal{I} \not\models^d F'$ for each $F' \in \rho_F \setminus \{F\}$. Since $k \models^d \rho_F$, we get $\mathcal{I} \models^d F$. ☐

Based on the above, we can now deal with step (I) discussed previously.

Theorem 1. *Let $\tau \in \mathsf{types}(K)$, let $\rho \in 2^{\mathbb{F}}$ be a disjunctive f-subquery, and let* KQ *be a d-complete kq-table. Then $\tau \models^d \rho$ iff for each knot $k \in K$ with root τ and each k-hit h of* KQ*, we have $h \cap \rho \neq \emptyset$.*

Proof. Suppose $\tau \models^d \rho$ but there exists a knot $k \in K$ with root τ and a k-hit h of KQ such that $h \cap \rho = \emptyset$. By Lemma 1 above, we have $\mathcal{I} \in \mathfrak{T}(k)$ such that $\mathcal{I} \not\models^d F$ for each $F \in \mathbb{F} \setminus h$, hence $\mathcal{I} \not\models^d F$ for each $F \in \rho$. This contradicts $\tau \models^d \rho$.

Suppose $\tau \not\models^d \rho$ but for each knot $k \in K$ with root τ and each k-hit h of KQ, we have $h \cap \rho \neq \emptyset$. As $\tau \not\models^d \rho$, there exists some $\mathcal{I} \in \mathfrak{T}(\tau)$ such that $\mathcal{I} \not\models^d F$ for each $F \in \rho$. Let k be the knot at the root of $\mathcal{I}$ and consider the collection $C = \{\rho' \setminus \rho \mid (k, \rho') \in \mathsf{KQ}\}$. A simple consequence of the d-completeness of KQ is that $\emptyset \notin C$. Hence, some minimal hitting set of C exist. Take any such minimal hitting set h. Clearly, $h \cap \rho = \emptyset$ and h is a k-hit of KQ. Contradiction. ☐

Using the above Theorem 1, we can compute a d-complete tq-table TQ^d out of a d-complete kq-table KQ^d. The procedure which exploits the theorem is presented in Figure 5.

We now show how to obtain KQ^{d+1} from TQ^d. Intuitively, to make the step from d to $d + 1$, we must verify how each knot (r, S) in K can extend the mappings that exist in the β-trees of its children $(\alpha, \beta) \in S$, which are captured by the minimal hitting sets of TQ^d. This is formalized in the following notion of a ρ-*fulfilling* assignment.

Definition 17. (assignment, ρ-fulfillment) Given a knot $k = (r, S)$ from K, any function $g : S \to 2^{\mathbb{F}}$ is called a *(f-subquery) assignment for k*. We say g is *ρ-fulfilling*, where $\rho \in 2^{\mathbb{F}}$, if there exist some $(X, \Sigma) \in \rho$ and a mapping $\phi : X \to \{r\} \cup S$ satisfying the following conditions:

(M1) For each $x \in X$ with $\phi(x) = r$, we have $\{A|A(x) \in q\} \subseteq r$.

(M2) If $R(x, y) \in q$ is an atom with $\phi(x) = r$, then $\phi(y) \in S$.

(M3) If y is open but not free in X, then $\phi(y) = r$.

(M4) For each $s = (\alpha, \beta)$ in S, there exist $\{(X_s^1, \Sigma_s^1), \ldots, (X_s^m, \Sigma_s^m)\} \subseteq g(s)$ such that:

- $\{x \in X \mid \phi(x) = s\} = \bigcup_{1 \leq i \leq m}(X_s^i)$,
- for every $y \in X_s^i$ that is free in X, (*i*) $\Sigma(y) \subseteq \alpha$, and (*ii*) if y is free in X_s^i, $\Sigma(y) \subseteq \Sigma_s^i(y)$.
- if $y \in X_s^i$ is open in X_s^i, then, for each $R \in \mathsf{back}(X_s^i, y)$, there is some $T \sqsubseteq^* R$ such that: (*i*) $T \in \alpha$, and (*ii*) if additionally y is free in X_s^i, then $T \in \Sigma_s^i(y)$.

Note that if g is ρ-fulfilling, then every assignment g' that contains g, i.e., with $g'(s) \supseteq g(s)$ for all $s \in S$, is ρ'-fulfilling for every $\rho' \supseteq \rho$.

Example 6. For the knot $k_1 = (\tau_{B,C}, \{s_1, s_2\})$ in Figure 2, where $s_1 = (\{T\}, \tau_A)$ and $s_2 = (\{P, T\}, \tau_{A,B,C})\})$, each assignment g with $g(s_2) = \{F_4\}$ is $\{F_6\}$-fulfilling, where as above $F_4 = (\{x_4\}, \{(x_4, \{T\})\})$ and $F_6 = (\{x_3, x_4\}, \emptyset)$. This is witnessed by the mapping $\phi(x_3) = \tau_{B,C}$ and $\phi(x_4) = s_2$, which satisfies the conditions M1 to M4 (for M4, consider $\{(X_{s_2}^1, \Sigma_{s_2}^1)\} \subseteq g(s_2)$ where $(X_{s_2}^1, \Sigma_{s_2}^1) = F_4$; $y = x_4$ is not free in $\{x_3, x_4\}$, but in $X_{s_2}^1 = \{x_4\}$). The same assignment is also $\{F_4\}$-fulfilling; to see this, simply set $\phi(x_4) = s_2$.

Intuitively, for a knot $k = (r, S)$ from K, a ρ-fulfilling assignment g witnesses the existence of a rooted prematch for ρ within depth $d + 1$ in an arbitrary $\mathcal{I} \in \mathfrak{T}(k)$, provided that, for each $s \in S$, the f-subqueries in $g(s)$ have rooted prematches in the subtree $\mathcal{I}_s$ of $\mathcal{I}$ rooted at s. Conversely, if $\mathcal{I} \models^{d+1} \rho$ for some $\mathcal{I}$, the assignment g that assigns to each $s \in S$ the set of (X, Σ) that are entailed at $\mathcal{I}_s$ is ρ-fulfilling. More precisely, we have:

Lemma 2. *Let $k = (r, S)$ be a knot in K and let $\rho \subseteq \mathbb{F}$. Further, let $\mathcal{I} \in \mathfrak{T}(k)$ and let g be an assignment such that $(X, \Sigma) \in g(s)$ iff $\mathcal{I}_s \models^d (X, \Sigma)$, for all $s \in S$. Then $\mathcal{I} \models^{d+1} \rho$ iff g is ρ-fulfilling.*

Proof. First we show ($\leftarrow$). If g is ρ-fulfilling, by assumption there is some $F_0 = (X_0, \Sigma_0) \in \rho$ and a mapping ϕ that satisfy M1 to M4 above. In particular, for each $s \in S$, there exists some set

$$\{(X_s^1, \Sigma_s^1), \ldots, (X_s^m, \Sigma_s^m)\} \subseteq g(s)$$

as described by M4. For each of these $F_s^i = (X_s^i, \Sigma_s^i)$, $\mathcal{I}_s \models^d (X_s^i, \Sigma_s^i)$ holds, so there is a rooted prematch π_s^i for F_s^i in $\mathcal{I}_s$.

We construct a rooted prematch for ρ in $\mathcal{I}$, by combining ϕ and the different π_s^i. The new mapping $\pi : X_0 \to \Delta^{\mathcal{I}}$ is defined as follows:

$$\pi(x) = \begin{cases} \mathsf{root}(\mathcal{I}) & \text{if } \phi(x) = r, \\ b_s^{\mathcal{I}} \cdot \pi_s^i(x) & \text{if } x \in X_s^i \text{ for some } s \text{ and } i \end{cases}$$

(recall that $b_s^{\mathcal{I}}$ is the unique node of $\mathcal{I}$ at depth 1 corresponding to s). Since $\{x \in X \mid \phi(x) = s\} = \bigcup_{1 \leq i \leq m}(X_s^i)$, π is well defined and total. It only remains to show that π is a rooted prematch for F_0 in $\mathcal{I}$.

1. Consider any $A(x) \in q$. If $x \in X_s^i$ for some s and i, then $\pi_s^i(x) \in A^{\mathcal{I}}$ because π_s^i is a rooted prematch, and hence $\pi(x) \in A^{\mathcal{I}}$. Otherwise, $\phi(x) = r$ and then M1 implies $A \in r$ and $\mathsf{root}(\mathcal{I}) \in A^{\mathcal{I}}$. Hence RP1 holds.

2. To show RP2, consider a pair $x, y \in X_0$ such that $R(x, y) \in q$. The following cases are possible:

   - If $x \in X_s^i$ for some s, then $y \in X_s^i$ (due to the closure properties of the set X_s^i). Since π_s^i is a rooted prematch, $\pi_s^i(y)$ is an R-successor of $\pi_s^i(x)$ in $\mathcal{I}_s$, and hence $\pi(y)$ is an R-successor of $\pi(x)$ in $\mathcal{I}$.
   - If $\phi(x) = r$, then by M2 we have $\phi(y) = s$ for some $s = (\alpha, \beta) \in S$. Also, by M4, $y \in X_s^i$ for some i and, as $x \notin X_s^i$, we have $R \in \mathsf{back}(X_s^i, y)$, and by the last item of M4, there is some $T \in \alpha$ such that $T \sqsubseteq^* R$ and, additionally, $T \in \Sigma_s^i(y)$ whenever y is free in X_s^i. Since π^i is a rooted prematch for F_s^i in $\mathcal{I}_s$, it satisfies RP3 and RP4. This implies that either $\pi_s^i(y)$ is the root of $\mathcal{I}_s$, or y is free in X_s^i. In the former case, $\pi(y)$ is an R-successor of $\pi(x)$ as desired. In the latter case, $T \in \Sigma_s^i(y)$ and $\pi_s^i(y)$ is a T-successor of the root of $\mathcal{I}_s$ by RP4, which also implies that $\pi(y)$ is an R-successor of $\pi(x)$.

3. RP3 follows directly from M3.

4. To show RP4, consider any y that is free in X_0 and an arbitrary $R \in \Sigma(y)$. There are two cases:

   - If $\phi(y) = r$, then $\pi(y) = \mathsf{root}(\mathcal{I})$ and RP4 holds.
   - If $y \in X_s^i$ for some s and i, then by the second item of M4, we have $R \in \alpha$ and either (*i*) y is open but not free in X_s^i, or (*ii*) $R \in \Sigma_s^i(y)$. Since π_s^i is a rooted prematch for F_s^i, it satisfies Definition 12. In case (*i*), RP3 implies that $\pi_s^i(y)$ is the root of $\mathcal{I}_s$, and hence an R-successor of $\mathsf{root}(\mathcal{I})$ in $\mathcal{I}$ as desired. In case (*ii*), RP4 implies that $\pi_s^i(y)$ is either the root of $\mathcal{I}_s$ as above, or an R-successor of it. As $R \in \alpha$, again in both cases $\pi_s^i(y)$ is an R-successor of $\mathsf{root}(\mathcal{I})$ in $\mathcal{I}$ as desired.

This shows that π is a rooted prematch for F_0 and hence $\mathcal{I} \models \rho$. Furthermore, since for each $x \in X_0$ the length of $\pi(x)$ is 0 if $\phi(x) = r$ and $\pi_s(x) + 1$ otherwise, and the length of $\pi_s(x)$ is bounded by d, we have $\mathcal{I} \models^{d+1} \rho$.

Now, to show ($\rightarrow$), we assume $\mathcal{I} \models^{d+1} \rho$, and that g is an assignment with $(X, \Sigma) \in g(s)$ for each $(X, \Sigma) \in \mathbb{F}$ and each $s \in S$ such that $\mathcal{I}_s \models^d (X, \Sigma)$. To see that g is ρ-fulfilling, we start by observing that, by assumption, there is a $(X, \Sigma) \in \rho$ and a rooted prematch π for (X, Σ) in $\mathcal{I}$. For each $s \in S$, let X_s contain all variables $x \in X$ such that $\pi(x)$ is in the tree $\mathcal{I}_s$. We partition X_s into sets of variables $X_s^1, \ldots, X_s^m$ that are connected in q. We define a function Σ_s^i that maps each free $y \in X_s^i$ to a set of transitive roles as follows: if $\pi(y)$ is the root of $\mathcal{I}_s$, then $\Sigma_s^i(y) = \mathbf{R}^+(\mathcal{K})$. Otherwise, $\Sigma_s^i(y) = \{R \in \mathbf{R}^+(\mathcal{K}) \mid \pi(y)$ is an R-successor of the root of $\mathcal{I}_s\}$. Clearly, each X_s^i is closed under the rules (a.i) and (a.ii) of Definition 11. Hence, to see that each (X_s^i, Σ_s^i) is an f-subquery, it suffices to observe that, since π is a rooted prematch, $\pi(y)$ is an R-successor of $\mathsf{root}(\mathcal{I})$ for each $R \in \mathsf{back}(X_s^i, y)$, and hence condition (b) also holds.

It is also easy to see that, for each s and each i, $\mathcal{I}_s \models^d (X_s^i, \Sigma_s^i)$ (simply restrict π to the corresponding variables to obtain a rooted prematch in $\mathcal{I}_s$). So, by our assumption about g, $(X_s^i, \Sigma_s^i) \in g(s)$.

Now we can define a mapping $\phi : X \rightarrow \{r\} \cup S$ that witnesses that g is ρ-fulfilling as $\phi(x) = r$ if $\pi(x)$ is the root of $\mathcal{I}$, and $\phi(x) = s$ if $x \in X_s$. It is straightforward to verify that ϕ satisfies M1 to M3 in Definition 17. For M4, we can use for each $s \in S$ the $(X_s^1, \Sigma_s^1), \ldots, (X_s^m, \Sigma_s^m)$ defined above, since they are in $g(s)$. Then the first item is trivial; the other two can be verified as follows:

- Consider any $y \in X^i_s$ that is free in X, and any $R \in \Sigma(y)$. Since π is a rooted prematch and $\pi(y) \neq \mathsf{root}(\mathcal{I})$, by RP4, $\pi(y)$ is an R-successor of $\mathsf{root}(\mathcal{I})$. This implies that, if $s = (\alpha, \beta)$, $R \in \alpha$ and either $\pi(y) = b^{\mathcal{I}}_s$ or $\pi(y)$ is an R-successor of $b^{\mathcal{I}}_s$. If y is also free in X^i_s then, in both cases, $R \in \Sigma^i_s(y)$ by construction of Σ^i_s.

- Consider any $y \in X^i_s$ that is open in X^i_s, and an $R \in \mathsf{back}(X^i_s, y)$ such that $R(x, y) \in q$. Since π is a rooted prematch and $x \notin X_s$, either (a) $\pi(x) = \mathsf{root}(\mathcal{I})$ or (b) $x \notin X$. In case (a), by RP2, $\pi(y)$ is a T-successor of $\mathsf{root}(\mathcal{I})$. In case (b), y is open in X and, moreover, free in X (as y open but not free would imply $\pi(y) = \mathsf{root}(\mathcal{I})$, contradicting $y \in X_s$). Hence, there is some $T \sqsubseteq^* R$ with $T \in \Sigma(y)$ and, by RP4, we also have $\pi(y)$ a T-successor of $\mathsf{root}(\mathcal{I})$. In both cases (a) and (b), it thus follows $T \in \alpha$, where $s = (\alpha, \beta)$. It also follows that either $\pi(y) = b^{\mathcal{I}}_s$, or $\pi(y)$ is a T-successor of $b^{\mathcal{I}}_s$ and T is transitive. If y is free in X^i_s, then by construction in both cases $T \in \Sigma^i_s$. □

The step from TQ^d to KQ^{d+1} computes the f-subqueries ρ for which the τ-hits of TQ^d are ρ-fulfilling.

Theorem 2. *Suppose* TQ *is a d-complete tq-table, $\rho \subseteq \mathbb{F}$ is a disjunctive f-subquery, and $k = (r, S)$ is a knot in K. Furthermore, let C be the set of assignments for k that map each $(\alpha, \beta) \in S$ to a β-hit of* TQ. *Then $k \models^{d+1} \rho$ iff every assignment $g \in C$ is ρ-fulfilling.*

Proof. ($\rightarrow$) Suppose $k \models^{d+1} \rho$. Consider an arbitrary assignment $g \in C$. By assumption, for each $s = (\alpha, \beta) \in S$, $g(s)$ is a β-hit of TQ. Hence, by Lemma 1, there is a tree $\mathcal{I}_s \in \mathfrak{T}(\beta)$ that satisfies exactly the f-subqueries in $g(s)$. Let $\mathcal{I} \in \mathfrak{T}(k)$ be the tree that coincides with all these $\mathcal{I}_s$. As $\mathcal{I} \models^{d+1} \rho$, then by Lemma 2, g is ρ-fulfilling.

($\leftarrow$). Now assume that each $g \in C$ is ρ-fulfilling, and consider an arbitrary $\mathcal{I} \in \mathfrak{T}(k)$. For each $s = (\alpha, \beta) \in S$, let $\mathbb{F}_s$ be the set of all f-subqueries F such that $\mathcal{I}_s \models^d F$, and let g' be the assignment such that $g'(s) = \mathbb{F}_s$ for all $s \in S$. Then, $\mathbb{F}_s \cap \rho' \neq \emptyset$ must hold for each $(\tau, \rho) \in \mathsf{TQ}$ such that $\tau = \beta$; hence, there exists some β-hit h_s of TQ such that $h_s \subseteq \mathbb{F}_s$. By assumption, there exists some $g \in C$ such that $g(s) = h_s$ for all $s \in S$. As g is ρ-fulfilling and g' contains g, also g' is ρ-fulfilling. Thus by Lemma 2, $\mathcal{I} \models^{d+1} \rho$. Hence, $k \models^{d+1} \rho$. □

Example 7. Reconsider the knot $k_1 = (\tau_{B,C}, \{s_1, s_2\})$ in Figure 2, where $s_1 = (\{T\}, \tau_A)$ and $s_2 = (\{P, T\}, \tau_{A,B,C})\})$, and the tq-table TQ^0 in Example 5. As for s_1, the single τ_A-hit of TQ^0 is $\emptyset$ (by minimality, as there is no entry for τ_A in TQ^0), and for s_2, the single $\tau_{A,B,C}$-hit of TQ^0 is $\{F_4, F_5\}$. Hence, the set of assignments C consists of g where $g(s_1) = \emptyset$ and $g(s_2) = \{F_4, F_5\}$. Since we know from Example 6 that g is $\{F_4\}$-fulfilling and $\{F_6\}$-fulfilling, it follows that $k_1 \models^1 \rho$ for every disjunctive f-subquery ρ that includes either F_4 or F_6 (or both); in particular, $k_1 \models^1 \{F_4\}$ and $k_1 \models^1 \{F_6\}$, and thus $(k_1, \{F_4\})$ and $(k_1, \{F_6\})$ are included in KQ^1.

So far, we have only considered singleton disjunctive f-subqueries in tq- and kq-tables, but not always complete such tables can be derived where only singleton f-subqueries occur. For example, if we continue the computation above, we would eventually compute a 2-complete tq-table TQ^2 such that each of its τ_{ABC}-hits contains either $F_6 x X <$ or F_2, and one can infer that $k_1 \models^3 \{F_1, F_2\}$ although neither $k_1 \models^3 \{F_2\}$ nor $k_1 \models^3 \{F_2\}$ holds.

The algorithm based on the Theorem 2, $\mathsf{compute\_TQ}$, is shown in Figure 6. Using the two algorithms presented so far, we can compute a d-complete TQ for any $d > 0$. A 0-complete tq-table TQ^0 can be constructed as described above. Due to monotonicity, we can obtain *one* TQ that is complete for any $d \in N$, the computation reaches a fixpoint. And in fact, this happens within finitely many steps.

```
Algorithm 3: KQ_from_TQ
  Data: a $d$-complete tq-table $\mathsf{TQ}$
  Result: a $d$+1–complete kq-table $\mathsf{KQ}$
  begin
      $R := \emptyset$;
      forall $k \in K$ do
          $C := \emptyset$;
          Add to $C$ each $g : S \to 2^{\mathbb{F}}$ such that for each $(\alpha, \beta) \in S$, $g((\alpha, \beta))$ is a $\beta$-hit of $\mathsf{TQ}$;
          forall $\rho \subseteq \mathbb{F}$ do
              if each $g \in C$ is $\rho$-fulfilling then
                  $R := R \cup \{(k, \rho)\}$;
      return $R$
  end
```

Figure 6: From type-query tables to knot-query tables (for the knot set K and the query q).

Proposition 5. *For each type τ, if $\tau \models \rho$, then there exists some $d \in \mathbb{N}$ such that $\tau \models^d \rho$.*

Proof. To give a bound d, we construct a tree of interpretations which captures parts of trees in $\mathfrak{T}(\tau)$ that are relevant for the mappings of ρ. For an integer $n \geq 0$, let $\mathcal{I}_{\uparrow n}$ be the restriction of a tree-shaped $\mathcal{I}$ up to depth n. Define a tree $T = (V, E)$ where

(i) the vertex set is $V = \{\mathcal{I}_{\uparrow n} \mid \mathcal{I} \in \mathfrak{T}(\tau) \wedge n \geq 0\}$, and

(ii) the child relation is $E = \{\langle \mathcal{I}_{\uparrow n}, \mathcal{I}_{\uparrow n+1} \rangle \mid \mathcal{I} \in \mathfrak{T}(\tau) \wedge n \geq 0\}$.

Intuitively, each pair in E represents an expansion of the levels 0, 1,. . . , $n - 1$ of $\mathcal{I}$ by another level using the knots in K. Hence each path in T corresponds to an interpretation in $\mathfrak{T}(\tau)$. Observe that T is finitely branching. Consider now the set P of all nodes $\mathcal{I}_{\uparrow n} \in V$ such that $\mathcal{I}_{\uparrow n} \models \rho$ and $\mathcal{I}_{\uparrow n-1} \not\models \rho$ (the latter in case where $n > 0$). As $\tau \models \rho$, by construction of T each path in it contains some node from P. Let T' result from T by removing all successors of nodes in P. Since T' does not have infinite branches and is finitely branching, by König's Lemma T' is finite. Hence for each $\mathcal{I} \in \mathfrak{T}(\tau)$ the match for ρ occurs within finite depth d, where d is the length of the longest branch in T'. ☐

Naturally, the question is which d witnesses $\tau \models \rho$ for sure. As the complexity analysis of our algorithm will reveal (see Section 6), a number double exponential in the size of the query and the knowledge base is sufficient.

5.3 Query Entailment over full KBs

Assume an arbitrary ABox $\mathcal{A}$. We now show how to use the knot set K and a complete type-query table to answer queries over the full knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$. The underlying principle is the same as above, but we need some technical machinery. Roughly speaking, the idea is to reduce answering q over $\mathcal{K}$ to answering q over a set of ABoxes. To this end, we construct the set $\mathsf{exp}(\mathcal{A}, \mathsf{TQ})$ which contains ABoxes obtained by expanding completions of $\mathcal{A}$ with one layer of knots (resulting, intuitively, in a forest of depth

```
Algorithm 4: compute_TQ
  Result: a complete tq-table TQ
  begin
      Construct a $0$-complete tq-table $\mathsf{TQ}^0$ for $K$ and $q$;
      $d = 0$;
      repeat
          $\mathsf{KQ}^{d+1} := \mathsf{KQ\_from\_TQ}(\mathsf{TQ}^d)$;
          $\mathsf{TQ}^{d+1} := \mathsf{TQ\_from\_KQ}(\mathsf{KQ}^{d+1})$;
          $d := d + 1$;
      until $\mathsf{TQ}^{d-1} \neq \mathsf{TQ}^d$ ;
      return $R$
  end
```

Figure 7: Computing a complete type-query table (for the knot set K and the query q).

1), and then attaching to each leaf ABoxes representing f-subqueries from TQ. The answer to q over $\mathcal{K}$ is then given by the tuples that belong to the answer of q over $\mathcal{A}'$ for every $\mathcal{A}' \in \mathsf{exp}(\mathcal{A}, \mathsf{TQ})$.

To query ABoxes, we define the following:

Definition 18 (querying ABoxes)**.** Given any ABox $\mathcal{A}'$, $\mathsf{ans}(q, \mathcal{A}')$ consists of individual tuples $\vec{c}$ such that $\vec{c} \in \mathsf{ans}(q, \mathcal{A}')$ iff $\vec{c} \in \mathsf{ans}(q, \mathcal{I})$, where $\mathcal{I}$ is the interpretation such that:

(a) $\Delta^{\mathcal{I}} = \mathbf{I}(\mathcal{A}')$,

(b) for each $a \in \mathbf{I}(\mathcal{A}')$, we have $a^{\mathcal{I}} = a$,

(c) for each atomic A, $a^{\mathcal{I}} \in A^{\mathcal{I}}$ iff $a : A \in \mathcal{A}'$, and

(d) for each role R, $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ iff $\langle a, b \rangle : R \in \mathcal{A}'$.

The interpretation $\mathcal{I}$ *represents* $\mathcal{A}'$.

We describe how $\mathsf{exp}(\mathcal{A}, \mathsf{TQ})$ is constructed, and the first step is to represent f-subqueries as ABoxes.

Definition 19. Given an individual b and an f-subquery $F = (X, \Sigma) \in \mathbb{F}$, let $\mathsf{abox}(b, F)$ be an ABox consisting of:

(a) $b : C$ for each open variable $x \in X$ and $C(x) \in q$.

(b) $b_{F,x} : C$ for each closed (i.e., non-open) variable $x \in X$ and $C(x) \in q$.

(c) $\langle b, b_{F,y} \rangle : R$ for each pair of an open $x \in X$ and closed $y \in X$ with $R(x, y) \in q$.

(d) $\langle b_{F,x}, b_{F,y} \rangle : R$ for each pair of closed variables $x, y \in X$ with $R(x, y) \in q$.

Note that $\mathsf{abox}(b, F)$ does not depend on Σ in $F = (X, \Sigma)$; indeed, Σ only serves for the back-propagation of subqueries which does not play a role here. We now make one step further and construct an ABox using a knot and an assignment, where the queries given by the latter induce a set of ABoxes as above.

Definition 20. Given an individual a, a knot $k = (r, S)$ and an assignment g for k, let $\mathsf{abox}(a, k, g) = \mathcal{A}_1 \cup \mathcal{A}_2$, where:

(a) The ABox $\mathcal{A}_1$ consists of:

- (i) $a : C$ for each $C \in r$;
- (ii) $\langle a, a_s^{k,g} \rangle : R$ for each $s = (\alpha, \beta)$ in S and $R \in \alpha$;
- (iii) $a_s^{k,g} : C$ for each $s = (\alpha, \beta)$ in S and $C \in \beta$;

(b) The ABox $\mathcal{A}_2$ is defined as $\mathcal{A}_2 = \bigcup_{s \in S} \bigcup_{F \in g(s)} \mathsf{abox}(a_s^{k,g}, F)$.

Before we present the algorithm, we look at the correspondence between constructed ABoxes and tree-shaped interpretations generated by knots.

Definition 21. Given a knot k and an assignment g for $k = (\tau, S)$, we say an interpretation $\mathcal{I} \in \mathfrak{T}(k)$ *represents k and g*, if for each $e \in \Delta^{\mathcal{I}}$ at depth 1, it holds that $\{F \in \mathbb{F} \mid \mathcal{I}|e \models F\} = g(s)$ where $e = b_s^{\mathcal{I}}$ (the unique element at depth 1 identified by the leaf $s \in S$ of k).

Informally, the above means that if we look at the subtree rooted at a child of the root of $\mathcal{I}$, then the set of queries for which it has a prematch is exactly the one given by the assignment.

A straightforward application of Lemma 1 yields the following:

Proposition 6. *Suppose $k = (r, S)$ is a knot in K and g is an assignment for k such that $g(\alpha, \beta)$ is a β-hit of a complete tq-table TQ for each $(\alpha, \beta) \in S$. Then some $\mathcal{I} \in \mathfrak{T}(k)$ exists that represents k and g.*

Intuitively, if $\mathcal{I} \in \mathfrak{T}(k)$ represents k and g, then $\mathsf{abox}(a, k, g)$ can be viewed as "compact" structure that is equivalent to $\mathcal{I}$ w.r.t. query prematches. We are now ready to define the ABox expansions.

Definition 22 (ABox expansions)**.** Given a complete tq-table TQ, a TQ-*expansion* of $\mathcal{A}$ is any ABox

$$\mathcal{A}' = \mathcal{A}_c \cup \bigcup_{a \in \mathbf{I}(\mathcal{A})} \mathcal{A}_a,$$

where $\mathcal{A}_c \in \mathsf{comp}(\mathcal{A})$ and, for each individual $a \in \mathbf{I}(\mathcal{A})$, $\mathcal{A}_a = \mathsf{abox}(a, k_a, g_a)$ for some $k_a = (r, S)$ in K with root type $\mathcal{A}_c(a)$ and assignment g_a for k_a such that $g_a(\alpha, \beta)$ is a β-hit of TQ for each $(\alpha, \beta) \in S$. We denote with $\mathsf{exp}(\mathcal{A}, \mathsf{TQ})$ the set of all TQ-expansions of $\mathcal{A}$.

Now we can formally state the main result of this section, which shows that we can reduce CQ answering over $\mathcal{K}$ to CQ answering over expanded ABoxes.

Theorem 3 (Main result)**.** *If TQ is a complete tq-table, then*

$$\mathsf{ans}(q, \mathcal{K}) = \bigcap_{\mathcal{A}' \in \mathsf{exp}(\mathcal{A}, \mathsf{TQ})} \mathsf{ans}(q, \mathcal{A}').$$

Proof. "$\rightarrow$". Suppose $\vec{c} \in \mathsf{ans}(q, \mathcal{K})$, where $\vec{c} = \langle c_1, \ldots, c_n \rangle$, and consider any $\mathcal{A}' \in \mathsf{exp}(\mathcal{A}, \mathsf{TQ})$. By definition, $\mathcal{A}' = \mathcal{A}_c \cup \bigcup_{a \in \mathbf{I}(\mathcal{A})} \mathcal{A}_a$, where $\mathcal{A}_c \in \mathsf{comp}(\mathcal{A})$ and, for each individual $a \in \mathbf{I}(\mathcal{A})$, $\mathcal{A}_a = \mathsf{abox}(a, k_a, g_a)$ for some $k_a = (r, S)$ in K with root type $\mathcal{A}_c(a)$, and some assignment g_a for k_a such that $g_a(\alpha, \beta)$ is a β-hit of TQ for each $(\alpha, \beta) \in S$.

```
Algorithm 5: computeAnswers
  Data: KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, CQ $q$
  Result: $\mathsf{ans}(q, \mathcal{K})$
  begin
      Compute a $C^{\mathcal{T}}_{\mathcal{A}}$-complete knot set $K$ for $\mathcal{T}$;
      Compute a complete tq-table $\mathsf{TQ}$ for $q$ from $K$;
      Compute the set $\mathsf{exp}(\mathcal{A}, \mathsf{TQ})$ of $\mathsf{TQ}$-expansions of the ABox $\mathcal{A}$;
      Let $R := \bigcap_{\mathcal{A}' \in \mathsf{exp}(\mathcal{A},\mathsf{TQ})} \mathsf{ans}(q, \mathcal{A}')$;
      return $R$
  end
```

Figure 8: Computing the answers to the query q over the KB $\mathcal{K}$.

We simply "expand" $\mathcal{A}_c$ to a full forest-base $\mathcal{I} \in \mathfrak{F}_K(\mathcal{A})$ for $\mathcal{K}$. We do this by taking an interpretation $\mathcal{I}_c$ representing $\mathcal{A}_c$ and, for each individual a, attaching to $a^{\mathcal{I}}$ a tree $\mathcal{I}_a$ that represents k_a and g_a. By assumption, q has some prematch π in $\mathcal{I}$ such that $\langle c_1^{\mathcal{I}}, \ldots, c_n^{\mathcal{I}} \rangle = \langle \pi(\vec{x}_1), \ldots, \pi(\vec{x}_n) \rangle$, where $\vec{x} = \langle x_1, \ldots, x_n \rangle$ are the answer variables of q. Consider a decomposition of π into several functions. Let $\pi_{\mathbf{I}}$ be the restriction of π to variables mapped to (the interpretation of) individuals. Furthermore, for each element e at depth 1 in $\mathcal{I}$, i.e., a child of a root node, let π_e be the restriction of π to elements that π maps in the subtree of $\mathcal{I}$ rooted at e. Clearly, π_e induces a (possibly empty) set of f-subqueries $Q \in 2^{\mathbb{F}}$ which all have a rooted prematch in the subtree of $\mathcal{I}$ rooted at e. Assume e is inside $\mathcal{I}_a$. Since $\mathcal{I}_a$ represents k_a and g_a, for all $F = (X, \Sigma) \in Q$ in the $\mathsf{abox}(e, F)$ we will have a prematch for the subquery of q induced by X. By composing $\pi_{\mathbf{I}}$ with the prematches for the latter subqueries of q, we can obtain a prematch π' for q in the interpretation $\mathcal{I}'$ representing $\mathcal{A}'$ such that $\langle c_1^{\mathcal{I}'}, \ldots, c_n^{\mathcal{I}'} \rangle = \langle \pi'(\vec{x}_1), \ldots, \pi'(\vec{x}_n) \rangle$. Hence, $\vec{c} \in \mathsf{ans}(q, \mathcal{A}')$.

"$\leftarrow$". Suppose $\vec{c} \in \mathsf{ans}(q, \mathcal{A}')$ for each $\mathcal{A}' \in \mathsf{exp}(\mathcal{A}, \mathsf{TQ})$. Let $\mathcal{I}$ be an arbitrary forest-base for $\mathcal{K}$. Due to $C^{\mathcal{T}}_{\mathcal{A}}$-completeness of K, we have $\mathcal{I} \in \mathfrak{F}_K(\mathcal{A})$, i.e., $\mathcal{I}$ can be constructed from some completion $\mathcal{A}_c$ of $\mathcal{A}$ and from knots in K. For each individual $a \in \mathbf{I}(\mathcal{A})$, let $k_a = (r_a, S_a)$ be the knot at the root of the subtree rooted at $a^{\mathcal{I}}$. Consider an assignment g'_a for k_a such that for each $s = (\alpha, \beta)$ in S_a, $g'_a(s)$ is the set of all f-subqueries that have a rooted prematch in the subtree of $\mathcal{I}$ rooted at e_s, where e_s is the child of $a^{\mathcal{I}}$ corresponding to the leaf s of k_a. Clearly, the subtree of $\mathcal{I}$ rooted at e_s is a β-tree. Due to completeness of TQ, there exists some β-hit h of TQ such that $h \subseteq g'(s)$. Thus we can define an assignment g_a for k_a such that for each $s = (\alpha, \beta)$ in S_a, $g_a(s)$ is a β-hit of TQ. Now consider an expansion $\mathcal{A}' \in \mathsf{exp}(\mathcal{A}, \mathsf{TQ})$ built from $\mathcal{A}_c$ and ABoxes $\mathsf{abox}(a, k_a, g_a)$ for each individual a where k_a and g_a are as described. Then the model $\mathcal{I}'$ that represents $\mathcal{A}'$ coincides with $\mathcal{I}$ on all individuals $a \in \mathbf{I}(\mathcal{A})$ and by Proposition 6, the subtree of $\mathcal{I}'$ rooted at a represents k_a and g_a. As $\vec{c} \in \mathsf{ans}(q, \mathcal{A}')$, there exists a prematch π' for q in $\mathcal{I}'$ such that $\langle c_1^{\mathcal{I}'}, \ldots, c_n^{\mathcal{I}'} \rangle = \langle \pi'(\vec{x}_1), \ldots, \pi'(\vec{x}_n) \rangle$. As $g_a(s) \subseteq g'_a(s)$ holds for all $a \in \mathbf{I}(\mathcal{A})$ and $s \in S$, it follows that q has a prematch π in $\mathcal{I}$ such that $\langle c_1^{\mathcal{I}}, \ldots, c_n^{\mathcal{I}} \rangle = \langle \pi(\vec{x}_1), \ldots, \pi(\vec{x}_n) \rangle$. $\square$

6 Computational Complexity

We analyze now the complexity of our algorithm for CQ answering over $\mathcal{SH}$ knowledge bases. Recall that the method consists of three main steps: (1) computing a $C^{\mathcal{T}}_{\mathcal{A}}$-complete knot set for an input KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, (2) computing a complete tq-table TQ for the computed knot set K and an input CQ q, and (3) collecting answers to q by traversing TQ-expansions of $\mathcal{A}$. The method is worst-case optimal for $\mathcal{SH}$, and, in fact, also for $\mathcal{ALCH}$. In order to see this, we first state the following theorem.

Theorem 4. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{SH}$ KB, let q be a conjunctive query, and let $\mathbb{F}$ be the set of f-subqueries of q. Then, given a tuple $\vec{c}$ of individuals, deciding whether $\vec{c} \in \mathsf{ans}(q, \mathcal{K})$ is feasible in time exponential in $|\mathcal{K}| + |q| + |\mathbb{F}|$.*

Proof. We will analyze the three steps of the procedure. To this end, let $c := |\mathsf{clos}(\mathcal{T})|$, and observe that the number of distinct types τ for $\mathcal{T}$ is bounded by $tm := 2^c$, while the number of distinct knots for $\mathcal{T}$ is bounded by $km := 2^c \cdot (2^c \cdot 2^c)^c = 2^{c+2c^2}$. Hence, the number of distinct types resp. knots is single exponential in $|\mathcal{K}|$.

For step 1, observe that a $C^{\mathcal{T}}_{\mathcal{A}}$-complete knot set K can be obtained in time exponential in $|\mathcal{K}|$ via the elimination algorithm from Figure 3. Indeed, constructing the set K in the first step is feasible in time exponential in $|\mathcal{T}|$. In the subsequent “fill-up” stage that closes K, the procedure may add only exponentially many knots, while in the final “clean-up” stage each removed knot cannot be introduced again.

For the step 2, we make sure that using the procedure from Figure 7 we can compute a complete type-query table TQ for $\mathcal{K}$ and q in time exponential in $s := |\mathcal{K}| + |q| + |\mathbb{F}|$. This follows from the next observations:

i) At each iteration, by construction, the algorithm computes a $d{+}1$-complete tq-table TQ^{d+1} (via a $d{+}1$-complete kq-table KQ^{d+1}) from a d-complete tq-table TQ^d. Furthermore, the computed tables are “full” in the sense that *all* entailed disjunctive f-subqueries queries are included in the tables.[2] More precisely, for each d, we have $(\tau, \rho) \in \mathsf{TQ}^d$ iff $\tau \models_d \rho$. Since $\tau \models_d \rho$ implies $\tau \models_{d+1} \rho$, we get that the computation is monotonic, i.e., each computed TQ^{d+1} includes TQ^d.

ii) The largest possible tq-table for $\mathcal{T}$ and q is $\mathsf{clos}(\mathcal{T}) \times 2^{\mathbb{F}}$, which is clearly of size exponential in s. Hence, and given the monotonicity, the algorithm terminates within a number of steps that is exponential in s.

iii) Each iteration, which consists of a call to $\mathsf{KQ\_from\_TQ}$ and then a call to $\mathsf{TQ\_from\_KQ}$, takes time at most exponential in s. For the call to $\mathsf{KQ\_from\_TQ}$, pairs k, ρ of knots k and disjunctive f-subqueries ρ are traversed. The number of such pairs is exponential in s. Furthermore, for each pair (k, ρ) its inclusion in the resulting table is decided by checking the conditions prescribed in Theorem 1, and this takes time at most exponential in s. The call to $\mathsf{TQ\_from\_KQ}$ is analogous: there are at most exponentially (in s) many pairs (τ, ρ) of types and disjunctive f-subqueries, and the inclusion test via the conditions in Theorem 2 is also feasible in exponential time.

For the final step 3, which is based on Theorem 3, note that the number of TQ-expansion of $\mathcal{A}$ is the number of ways of choosing a completion $\mathcal{A}_c$ of $\mathcal{A}$, expanding $\mathcal{A}_c$ for each individual a of $\mathcal{A}$ with a knot k and then choosing a set of f-subqueries for each leaf of the resulting forest (yielding an assignment g); this is again bounded by an exponential in s. Finally, checking whether $\vec{c} \in \mathsf{ans}(q, \mathcal{A}')$ is true for a given expansion $\mathcal{A}' \in \mathsf{exp}(\mathcal{A}, \mathsf{TQ})$ is also feasible in time exponential in s. $\square$

We can now easily infer the upper bound for the query answering problem in $\mathcal{SH}$. Indeed, for a given CQ q, the size of the set $\mathbb{F}$ of f-forward subqueries of q is bounded by $2^{\mathbf{V}(q)}$, which is exponential in $|q|$. Therefore, by the above theorem, the procedure can be run in time double exponential in the size of the input KB $\mathcal{K}$ and the query q. This is worst-case optimal due to the 2EXPTIME-hardness of the problem, which was shown in [10].

[2] In fact, only the subset minimal subqueries would need to be stored. However, the worst case complexity remains unchanged.

Theorem 5. *Given a KB $\mathcal{K}$ in $\mathcal{SH}$, a CQ q and a tuple of individuals $\vec{c}$, deciding whether $\vec{c} \in \mathsf{ans}(q, \mathcal{K})$ is* 2EXPTIME*-complete.*

We finally note that computing $\mathsf{ans}(\mathcal{K}, q)$ for an $\mathcal{SH}$ KB $\mathcal{K}$ and a query q is also feasible in double exponential in $|\mathcal{K}| + |q|$. This is because the number of candidate answer tuples $\vec{c}$ is only exponential in $|\mathcal{K}|$.

6.1 Syntactic Restrictions

We discuss here some syntactic restrictions to obtain classes of CQs for which query answering is feasible in single exponential time. To this end, it is sufficient to ensure that a query can be decomposed into only polynomially many f-subqueries; the complexity drop follows then from Theorem 4.

We assume an arbitrary $\mathcal{SH}$ KB $\mathcal{K}$, and define next some notions to measure the structural complexity a query (w.r.t. $\mathcal{K}$).

Definition 23 (fork degree, non-trivial forks)**.** For any query q, we define

$$\mathsf{R}^q_+(x) = \{x_n \mid R_1(x, x_1) \in q, R_2(x_1, x_2) \in q, \ldots, R_n(x_{n-1}, x_n) \in q \wedge n \geq 1\},$$

i.e., $\mathsf{R}^q_+(x)$ denotes the set of variables reachable from x in the query graph of q in one or more steps. Furthermore, let $\mathsf{R}^q_*(x) = \{x\} \cup \mathsf{R}^q_+(x)$ and, for any set X of variables, let $\mathsf{R}^q_+(X) = \bigcup_{x \in X} \mathsf{R}^q_+(x)$ and $\mathsf{R}^q_*(X) = \bigcup_{x \in X} \mathsf{R}^q_*(x)$.

A set $X \subseteq \mathbf{V}(q)$ is called a *fork set* (of q) if the following are true:

(a) for each $x \neq y \in X$, it holds that $y \notin \mathsf{R}^q_+(x)$ and $x \notin \mathsf{R}^q_+(y)$;

(b) the set $\mathsf{R}^q_*(X)$, i.e., the closure of X under reachable variables, induces a connected subquery of q;

(c) there exists no variable $x \in X$ such that, for some $y \in \mathbf{V}(q)$, we have $y \in \mathsf{R}^q_+(x)$ and $y \in \mathsf{R}^q_+(y)$, i.e., none of the variables in X reaches a cycle in q.

Then the *fork degree of* q, denoted $\mathsf{fd}(q)$, is defined as the size of the largest fork set of q.

The *number of non-trivial forks* in a query q is the number of variables $x \in \mathbf{V}(q)$ satisfying the following:

(a) there exist two atoms $R(z, x) \in q$ and $R'(z', x) \in q$ such that $z \neq z'$ and R' is not simple in $\mathcal{K}$,

(b) there exists no $y \in \mathbf{V}(q)$ such that $y \in \mathsf{R}^q_+(x)$ and $y \in \mathsf{R}^q_+(y)$.

Example 8. For the query q in Example 4, $X = \{x_1, x_5\}$ is the only fork set of q which contains more than one variable; any other such candidate fork set violates either condition (a) or condition (b). Hence, $\mathsf{fd}(q) = 2$.

Note that for fork sets X of size larger than one, each variable $x \in X$ has a common successor with some other variable $y \in X$ (in the previous example, x_1 has a common successor with x_5). Intuitively, the fork degree of q tells us how many "incomparable" variables we can pick so that they induce a connected acyclic subquery of q. Given this, we can formulate a syntactic condition ensuring lower complexity of query answering.

Theorem 6. *If $\mathcal{Q}$ is a class of CQs such that for any $q \in \mathcal{Q}$:*

(a) the number of non-trivial forks in q is bounded by some constant c,

(b) $\mathsf{fd}(q)$ *is bounded by c, and*

(c) for each pair $x, y \in \mathbf{V}(q)$, $|\{\, R \mid R(x, y) \in q \wedge R \text{ is not simple in } \mathcal{K}\}| \leq c$,

then the set $\mathbb{F}$ of f-subqueries of q is polynomial in $|q|$. Hence, answering a query $q \in \mathcal{Q}$ over the KB $\mathcal{K}$ is feasible in single exponential time in $|q| + |\mathcal{K}|$.

Proof. Consider an arbitrary f-subquery (X, Σ) of $q \in \mathcal{Q}$. By definition, X induces a connected acyclic subquery of q. Let M_x be the set of all $x \in X$ that have no predecessor in X, i.e., for which there exists no $R(z, x) \in q$ with $z \in X$. Clearly, M_x is a fork set of q (see Definition 23), and hence by (b) $|M_x| \leq c$. We get that the number of different M_x over all possible X is bounded by $|\mathbf{V}(q)|^c$, and is polynomial in $|q|$. It is not hard to see that given two f-subqueries (X_1, Σ_1) and (X_2, Σ_2) of q with $M_{x_1} = M_{x_2}$ we also have $X_1 = X_2$. Hence, the number of distinct X that can be chosen is bounded by $|\mathbf{V}(q)|^c$, and is polynomial in $|q|$.

We consider the possibilities of choosing Σ. Observe that Σ is defined only for the free variables of X. The number of variables $y \in X$ that are in free in X is bounded by $2{\cdot}c$ because of the bounded number of non-trivial forks in q (condition (a)). For each such y, we have $|\mathsf{back}(X, y)| \leq d$ for some constant d because of the conditions (a) and (c). Hence, the number of choices for $\Sigma(y)$ is bounded by $|\{R|R(x, y) \in q\}|^d$, and is thus polynomial in $|q|$.

The second part of the claim follows then from Theorem 4. ☐

Note that when computing the fork degree of a query, we do not ignore forks $R(x, y), R'(x', y)$ where $x \neq x'$ and R, R' are simple; the variables x and x' in this case are treated as incomparable. However, such forks are *simple* in the sense that they do not increase the number of distinct f-subqueries because (a.ii) of Definition 11 enforces that either both x and x' or neither x nor x' belong to a f-subquery. To deal with this, we eliminate such forks from the query.

Definition 24 (fork rewriting [23])**.** For a CQ q, a *fork rewriting* of q is a query obtained from q by exhaustively applying the following rule: if the query contains atoms $R(x, y)$ and $R'(x', y)$ where $x \neq x'$ and R, R' are simple, then replace every occurrence of x with x', By $\mathsf{fw}(q)$ we denote an arbitrary fork rewriting of q.

Example 9. Reconsider the query in Example 4 (Figure 4). Fork elimination is applicable to $P(x_1, x_6)$, $R(x_5, x_6)$; after that, no further elimination is possible and we have the result

$$\mathsf{fw}(q) = Q(x_2, x_3), R(x_3, x_4), A(x_4), C(x_4), B(x_5), C(x_5), T(x_5, x_2), P(x_5, x_6), R(x_5, x_6), Q(x_6, x_7), C(x_7).$$

Note that fork rewritings of q coincide up to a renaming of variables. We can now state the slightly relaxed conditions which diminish the impact of simple forks to the fork degree.

Theorem 7. *Let $\mathcal{Q}$ be a class of CQs such that for any $q \in \mathcal{Q}$:*

(a) the number of non-trivial forks in $\mathsf{fw}(q)$ *is bounded by some constant c,*

(b) $\mathsf{fd}(\mathsf{fw}(q))$ *is bounded by c, and*

(c) for each pair $x, y \in \mathbf{V}(q)$, $|\{R|R(x, y) \in q \wedge R \text{ is not simple in } \mathcal{K}\}| \leq c$.

Then deciding $\vec{c} \in$ $\mathsf{ans}(q, \mathcal{K})$ *for a given $q \in \mathcal{Q}$ and tuple $\vec{c}$, is* EXPTIME*-complete in $|q| + |\mathcal{K}|$.*

Proof. To prove the upper bound, by Theorem 4, it suffices to show that the number of f-subqueries of q is polynomial in $|q|$. As argued in the proof of Theorem 6, the conditions (a) and (c) ensure that the number of ways to choose Σ for an f-subquery (X, Σ) of q is polynomial in $|q|$.

The number of choices for X is also polynomial in $|q|$. To see this, for any conjunctive query q' define $\mathcal{X}_{q'} = \{X \mid \langle X, \Sigma\rangle$ is an f-subquery of $q'\}$, and observe that (*i*) $|\mathcal{X}_{\mathsf{fw}(q)}|$ is polynomial in $|q|$, and (*ii*) $|\mathcal{X}_q| = |\mathcal{X}_{\mathsf{fw}(q)}|$. The former follows from (b) and Theorem 6 (as $|\mathsf{fw}(q)| \leq |q|$). For the latter, note that a rewrite step in fork rewriting preserves the number of variable sets satisfying (a.ii) of Definition 11. More precisely, if q'' is obtained from q' by the rewrite rule in Definition 24, then $|\mathcal{X}_{q'}| = |\mathcal{X}_{q''}|$ (as easily seen by establishing a bijection from $\mathcal{X}_{q'}$ to $\mathcal{X}_{q''}$).

The lower bound easily follows from the EXPTIME-hardness of satisfiability testing in $\mathcal{ALC}$ [35]. ☐

Based on the above theorem we can obtain further query classes of lower computational complexity. In particular, the conditions (a-c) of Theorem 7 are satisfied for the class of queries that allow for simple roles only. Indeed, given such a query q, (a) and (c) are trivially satisfied. For (b), observe that for any variable x of $\mathsf{fw}(q)$ there are two possibilities. The variable x occurs in a cycle in the query graph of $\mathsf{fw}(q)$, and hence x is not included in any fork set and does not contribute to the fork degree. Alternatively, x and its successors induce a subquery of $\mathsf{fw}(q)$ whose graph is a tree. In this case, $\{x\}$ is the single fork set where x may occur. Therefore, $\mathsf{fw}(q) = 1$.

Importantly, the above can be generalized to the case where only a bounded number of atoms $R(x, y)$, where R is non-simple, occur in a query. This is a consequence of the next result, for which we use a more refined query complexity measure.

Definition 25 (counting transitive arcs)**.** For any query q, let $t(q)$ denote the number of all pairs of variables $x, y \in \mathbf{V}(q)$ such that:

(1) q contains some atom $R(x, y)$ where R is not simple in $\mathcal{K}$,

(2) q contains no atom $R'(x, y)$ where R' is simple in $\mathcal{K}$,

(3) y does not reach a cycle in the query graph of q, i.e., no $z \in \mathsf{R}^q_*(y)$ exists such that $z \in \mathsf{R}^q_+(z)$, and

(4) some variable $z \in \mathsf{R}^q_*(y)$ has more than one predecessor in q, i.e., $|\{u \mid R(u, z) \in q\}| > 1$.

Note that (3) eliminates pairs of variables that do not matter for the fork degree due to cyclicity (see (b) in Definition 23). Condition (4) refines this, by further eliminating cases where $\mathsf{R}^q_*(y)$ induces a query subgraph of q that is tree-shaped and disconnected to the remainder of the query graph of q. We remark that $t(q)$ can be easily computed.

Proposition 7. *For each CQ q it holds that* $\mathsf{fd}(\mathsf{fw}(q)) \leq t(\mathsf{fw}(q)) + 1 \leq t(q) + 1$.

The proof of this proposition, which does not give particular insight into the techniques of this section, is given in Appendix A.

Assume a query q and observe that the number of non-trivial forks in $\mathsf{fw}(q)$ is $\leq t(q)$. Indeed, due to the rewrite rule, for each variable y of $\mathsf{fw}(q)$ there exists at most one variable x such that $\{R \mid R(x, y) \in q\}$ contains a simple role. In other words, for each other variable $z \neq x$, all roles in $\{R \mid R(z, y) \in q\}$ must be non-simple. This means that if x is a variable counted in as a non-trivial fork (i.e., satisfies the conditions in Definition 23), then for x there exists at least one z such that the pair z, x is counted in $t(q)$ (i.e., z, x satisfy the conditions in Definition 25). Hence, and given Proposition 7, we reshape Theorem 7 as follows.

Theorem 8. *If $\mathcal{Q}$ is a class of CQs and c is a constant such that for any $q \in \mathcal{Q}$:*

(a) $t(q) \leq c$*, and*

(b) for each pair $x, y \in \mathbf{V}(q)$, $|\{ R \mid R(x, y) \in q \wedge R \text{ is not simple in } \mathcal{K}\}| \leq c$,

then deciding $\vec{d} \in \mathsf{ans}(q, \mathcal{K})$ *for a given* $q \in \mathcal{Q}$ *and tuple* $\vec{d}$*, is* EXPTIME*-complete in* $|q| + |\mathcal{K}|$.

Corollary 1. *(Full) query answering in $\mathcal{ALCH}$ is feasible in single exponential time in the size of the input.*

From the EXPTIME-hardness of consistency testing in $\mathcal{ALC}$ [35], it follows that the presented query answering procedure is also worst-case optimal for $\mathcal{ALCH}$.

6.2 Data Complexity and Encoding into Datalog

The query answering procedure presented here can be easily adjusted to be worst-case optimal in the data complexity. We analyze next the complexity of verifying $\vec{c} \in \mathsf{ans}(q, \langle \mathcal{T}, \mathcal{A} \rangle)$ where the terminology $\mathcal{T}$ and the query q are fixed, and only the ABox $\mathcal{A}$ with assertions over roles and atomic concepts is considered as an input. The CONP-hardness of the problem is well-known (see, e.g., [5] for more details), and we argue here that the method provides a tight CONP upper bound.

Proposition 8. *Algorithm* computeAnswers*, adapted to a nondeterministic version, runs in* CONP *data complexity.*

Proof. Fix a terminology $\mathcal{T}$ and a query q. As argued already, for a set Q of types for $\mathcal{T}$ we can obtain a Q-complete knot set K for $\mathcal{T}$ via the algorithm in Figure 3. To be capable of dealing with any possible input ABox $\mathcal{A}$, we set Q to the set of all possible types for $\mathcal{T}$, and compute a complete type-query TQ for K and q. Note the difference from the algorithm in Figure 8 where only the restricted set $C^{\mathcal{T}}_{\mathcal{A}}$ of types occurring in completions of an input ABox $\mathcal{A}$ is considered. In other words, we compute a knot set K and a tq-table TQ that are good for any possible input ABox $\mathcal{A}$.

Given K and TQ, the result follows from the fact that deciding $\vec{c} \notin \mathsf{ans}(q, \langle \mathcal{T}, \mathcal{A} \rangle)$ is feasible in nondeterministic polynomial time in the size of $|\mathcal{A}|$. Indeed, this can be done in a guess-and-check manner as follows:

- Build nondeterministically an expansion $\mathcal{A}' \in \mathsf{exp}(\mathcal{A}, \mathsf{TQ})$. More precisely, guess an ABox completion $\mathcal{A}_c$ and, for each individual $a \in \mathbf{I}(\mathcal{A})$, add $\mathsf{abox}(a, k_a, g_a)$ according to a nondeterministic choice of some knot $k_a = (r, S)$ from K with root type $\mathcal{A}_c(a)$, and some proper assignment g_a for k_a w.r.t. TQ (see Definition 22). Observe that since $\mathcal{T}$ and q are fixed, such an expansion $\mathcal{A}'$ can be nondeterministically computed in polynomial time in $|\mathcal{A}|$ ($\mathcal{A}_c$ has polynomial size then, and checking the conditions (a)-(g) of Definition 7 is simple; $\mathsf{abox}(a, k_a, g_a)$ has size bound by a constant, and only constantly many different $\mathsf{abox}(a, k_a, g_a)$ exist).

- Verify $\vec{c} \notin \mathsf{ans}(q, \mathcal{A}')$. There are $|\mathbf{I}(\mathcal{A}')|^{|\mathbf{V}(q)|}$ different candidate query mappings π for q in $\mathcal{A}'$. As $|\mathbf{V}(q)|$ is fixed and $|\mathbf{I}(\mathcal{A}')|$ is linear in $|\mathcal{A}|$, the number of such candidates is polynomial in $|\mathcal{A}|$. Testing whether π witnesses $\vec{c} \in \mathsf{ans}(q, \mathcal{A}')$ is also polynomial in $|\mathcal{A}|$. Hence, the verification step is feasible in polynomial time in $|\mathcal{A}|$. □

We get an analogous CONP result for answering varying CQs of small size (bounded by a constant) over a knowledge base $\mathcal{K}$ with static (fixed) TBox, if its compilation into a suitable Q-complete knot K set is available; this may be due to off-line pre-compilation, or to caching K after the first query. In this setting, the tq-table TQ is constructible in polynomial time (only constantly many subqueries exist), and deciding whether $\vec{c} \notin \mathsf{ans}(q, \langle \mathcal{T}, \mathcal{A}\rangle)$ is still feasible in NP. In fact, if also the ABox is fixed, the last step is feasible in polynomial time (only constantly many expansions $\mathcal{A}'$ of $\mathcal{A}$ and constantly many candidate mappings of $\mathbf{V}(q)$ into each $\mathcal{A}'$ exist, which can be easily traversed).

The guess-and-check procedure in the proof of Proposition 8 can easily be simulated in a disjunctive Datalog program [7], which consist of rules of the form

$$A_1 \vee \cdots \vee A_m \leftarrow B_1, \ldots, B_n \qquad m + n > 0 \tag{1}$$

where the A_i and B_j are function-free first-order atoms and each variable occurring in A_i also occurs in some B_j. The semantics of such a program P is given by the minimal (w.r.t. $\subseteq$) sets of ground (variable-free) atoms that are closed under the rules of P (called minimal models or answer sets); a ground atom A is a cautious consequence of P, if A occurs in all answer sets of P.

Using disjunctive rules, it is possible to generate the expansions $\mathcal{A}'$ of an input ABox $\mathcal{A}$ (with assertions over roles and atomic concepts only) in the answer sets of a ground program $P(\mathcal{A})$, such that $\mathsf{ans}(q, \langle \mathcal{T}, \mathcal{A}\rangle)$ corresponds to the set of cautious consequences $q(\vec{c})$ of $P(\mathcal{A})$. In more detail, viewing concepts and roles as predicates, and thus assertions $a : C$, $\langle a, b\rangle : R$ as atoms $C(a)$, $R(a, b)$, we can "guess" for each possible atom $C(a)$ resp. $R(a, b)$ with a rule $C(a) \vee \bar{C}(a) \leftarrow$ resp. $R(a, b) \vee \bar{R}(a, b) \leftarrow$, where $\bar{C}$ and $\bar{R}$ are fresh predicates, whether the atom belongs to an ABox completion $\mathcal{A}_c$ of $\mathcal{A}$, and ensure with rules of form (1) that for the so guessed $\mathcal{A}_c$ the conditions (a)-(g) of Definition 7 are satisfied. Furthermore, for each individual $a \in \mathbf{I}(\mathcal{A})$, we can guess some $\mathcal{A}_a = \mathsf{abox}(a, k_a, g_a)$ using a rule $\mathsf{abox}(a, k_1, g_1) \vee \cdots \vee \mathsf{abox}(a, k_{n_a}, g_{n_a}) \leftarrow$, where the $\mathsf{abox}(a, k_i, g_i)$ are all possible choices for a (here k and g are viewed as constant symbols). Facts for $\mathsf{abox}(a, k, g)$ according to Definitions 20 and 19 are generated with rules $C(a) \leftarrow \mathsf{abox}(a, k, g)$, $\quad R(a, a_s^{k,g}) \leftarrow \mathsf{abox}(a, k, g)$ etc. Finally, for each possible mapping π of $\mathbf{V}(q)$ into $\mathbf{I}(\mathcal{A}')$, the rule $q(\pi(x_1), \ldots, \pi(x_n)) \leftarrow P_1(\vec{y}_1\pi), \ldots, P_m(\vec{y}_m\pi)$ is added, given that $q = \{P_1(\vec{y_1}), \ldots, P_m(\vec{y_m})\}$ and q has answer variables $\vec{x} = \langle x_1, \ldots, x_n\rangle$; here $\vec{y}_i\pi$ denotes the substitution of $\pi(x_j)$ for x_j in the arguments $\vec{y_i}$ of P_i, for $j = 1, \ldots, n$.

Overall, the program $P(\mathcal{A})$ is constructible in polynomial time from $\mathcal{A}$, and since cautious inference from ground disjunctive Datalog programs is CONP-complete, this reduction is also worst-case optimal.

It is possible to lift this encoding to a fixed non-ground program $P_q^{\mathcal{T}}$ such that, for each input ABox $\mathcal{A}$, $\mathsf{ans}(q, \mathcal{A})$ corresponds to the cautious consequences $q(\vec{c})$ of $P_q^{\mathcal{T}} \cup \mathcal{A} \cup N_{\mathcal{A}}$, where $N_{\mathcal{A}}$ consists of facts $name_s^{k,g}(a, a_s^{k,g})$ and $name_{X,x}(a, a_{X,x})$ that introduce the new individuals $a_s^{k,g}$ and $a_{X,x}$ in the expansion $\mathcal{A}'$ (cf. Definitions 20 and 19) in the program; note that $N_{\mathcal{A}}$ has size linear in $|\mathbf{I}(\mathcal{A})|$ and is easily constructed from $\mathcal{A}$. Again, this is worst-case optimal.

We finally note that instead of disjunction, also (unstratified) negation may be used for the encoding. Thus, a range of reasoning engines for disjunctive/unstratified Datalog (e.g., DLV, smodels, clasp) can be used for implementation.

7 Discussion and Conclusion

The novel algorithm for CQ answering over knowledge bases in $\mathcal{SH}$ which we presented above has some nice features; it is worst case optimal for $\mathcal{SH}$ in general but also for important fragments including $\mathcal{ALC}$

and $\mathcal{ALCH}$, both with respect to the size of $\mathcal{K}$ plus q (the combined complexity) and the size of the ABox of $\mathcal{K}$ (the data complexity). Noticeably, the algorithm handles CQs with distinguished (output) variables directly and makes a customary grounding step unnecessary, in which an input query is reduced to (possibly exponentially many) Boolean queries, one for each possible output tuple $\vec{c}$. The underlying knot technique is different from previous query answering approaches yet not completely unrelated.

7.1 Related Work

Our knot technique can be seen as a special instance of the well-known mosaic method in Modal Logic (cf. [27, 26]) and is related to type elimination [32]. Mosaics are small "blocks" for building models that involve a bounded number of elements, and possibly infinite models are represented by finite sets of such blocks; types are roughly speaking "small mosaics" that involve at most two elements. The mosaic and type techniques have been applied in various contexts, including in DLs, cf. [24, 25, 34]. However, these works targeted deciding satisfiability of a knowledge base, i.e., existence of some model. We instead have applied and extended the mosaic technique to the more involved problem of CQ answering, which implicitly requires considering all models, or to find a suitable countermodel of the query. While in principle, one could use types as well, we feel that knots are better suited than types because of their more comprehensive representation of the local model structure.

Many different approaches for CQ answering have been developed that adapt known techniques for standard reasoning, including reduction to concept satisfiability (e.g., rolling up [17, 13]), resolution-based techniques [18], modified tableaux [21, 29], and tree-automata based algorithms [2, 3].

In the rolling up technique [17, 13, 16], CQ answering is reduced to deciding concept satisfiability by compiling the query into the knowledge base, using ideas of [6]. Roughly, in order to show $\mathcal{K} \not\models q$, one considers all possible ways in which a CQ q can be mapped to a canonical model of $\mathcal{K}$, i.e., all homomorphisms π of q into a tree- resp. forest-shaped model. Each such tree/forest mapping π is represented as a DL concept C_π, possibly in an extension $\mathcal{L}'$ of the DL $\mathcal{L}$ considered. Finally, one checks whether $\mathcal{K}' = \mathcal{K} \cup \{C_\pi \sqsubseteq \bot \mid \pi\}$ is satisfiable. Here, important aspects are that $\mathcal{K}'$ can be exponentially larger than $\mathcal{K}$, that the construction of C_π might not be easy, and that $\mathcal{L}'$ might have higher complexity than $\mathcal{L}$. For answering CQs and unions thereof in the DL $\mathcal{SHIQ}$, Glimm et al. [13, 16] applied rolling up in combination with extensive query rewriting, in order to arrive at a reduction to satisfiability in the DL $\mathcal{SHIQ}^\sqcap$, which extends $\mathcal{SHIQ}$ with role intersection. Satisfiability of $\mathcal{SHIQ}^\sqcap$ KBs is then decided using suitable tree automata. Overall, their algorithm runs in double exponential time and is thus worst-case optimal. However, since it behaves like traditional algorithms in absence of transitive roles, it is not single exponential for $\mathcal{ALCH}$. Lutz [22, 23] showed that a single exponential time bound is obtainable for $\mathcal{ALCH}$ using the rolling up and rewriting approach. We remark that with a similar approach as in [13, 16], Glimm et al. [14, 15] showed that answering CQs and unions thereof in the DL $\mathcal{SHOQ}$ is feasible in double exponential time (again this is worst-case optimal).

The resolution-based method by Hustadt et al. [19] is perhaps most closely related to ours. Similar as in our approach, their method first "compiles" the knowledge base and the query into a special form, and then exploits the possibility to answer the query by means of a disjunctive Datalog program. However, this is done on different grounds: the knot technique is model-theoretic in nature, while Hustadt et al.'s method is proof-theoretic, cleverly exploiting resolution and superposition machinery. Furthermore, the knot technique handles transitive roles in the query, which are not allowed in [19].

The tableaux method for satisfiability testing has been extended to CQ answering in [21, 29], with the aim to prove $\mathcal{K} \models q$ by showing that there is no countermodel to the query, i.e., no model of $\mathcal{K}$ in which q

is false. To this end, the tableaux blocking conditions are generalized to take the query q into account and depend on its size. A major drawback of the tableaux method is that, like for the resolution-based approach, handling transitive roles in the query seems to be difficult. It is not clear how to adapt the algorithms in [29], which work for queries with non-transitive roles in the DLs $\mathcal{SHIQ}$, $\mathcal{SHOQ}$, and $\mathcal{SHOI}$. Furthermore, the algorithms have nondeterministic triple exponential time complexity in general, which is far from worst-case optimal.

Finally, the tree-automata based approach to CQ answering [2, 3] exploits the tree- resp. forest-shaped model property of suitable DLs to solve the problem by combining automata for subproblems. It has been used to show that CQ answering for expressive DLs beyond $\mathcal{SHIQ}$ is feasible in double exponential time. To this end, forest-shaped interpretations of $\mathcal{K}$ are encoded into trees, and automata for recognizing models of $\mathcal{K}$ and of q are combined using intersection and complementation operations. Like the knots approach, tree-automata operate on small local parts of a model. However, while knots preserve the relational structure of these parts and can be extended to the needs of query answering, in tree-automata—which operate merely on strings—this structure is lost, and coding to the automata alphabet and state set is necessary. This in particular makes it hard to single out the impact of different components of $\mathcal{K}$ and q in the overall complexity, e.g. to derive results on data complexity. Furthermore, the algorithms in [2, 3] do not run in single exponential time for $\mathcal{ALCH}$.

For further comparison and discussion, see [11, 9].

7.2 Extensions and Further Work

The knot method we presented is extendible to richer DLs beyond $\mathcal{SH}$. Number restrictions can be accommodated by adapting the knot representation of knowledge bases. To this end, knots (τ, S) may be generalized such that S is a multi-set of types from $2^{\mathbf{R}(\mathcal{T})} \times \mathsf{clos}(\mathcal{T})$ that obeys numerical constraints. These constraints have to be suitably respected when composing knots, while no major change to the machinery of subqueries is necessary.

Inverse roles, which allow to relate an object to its parent and lead to upward arcs in tree-shaped models, can also be accommodated. To this end, the method of treeification can be applied to subqueries, which informally converts a query into ones whose query graphs are tree-shaped, by replacing atoms and renaming variables (see [9] for a detailed description). While this does not cause an exponential complexity increase in case of $\mathcal{SH}$, it causes one for $\mathcal{ALC}$, leading to a double exponential time algorithm for $\mathcal{ALCI}$; by Lutz's result [22], this is still worst-case optimal.

Also joint number restrictions and inverse roles can be handled by knots, but their possible interaction needs care and makes the extension of the technique more involved. On the other hand, it is unclear how to incorporate nominals into knots; the reason is that the forest-shaped model property gets lost.

We remark that [9] discusses a dual approach for CQ answering using knots: there, knots are associated with sets of subqueries and an elimination algorithm similar to the one in Figure 3 is used to test the existence of a model which falsifies each subquery at each knot. This is in contrast to the algorithm here, where the entailed disjunctions of subqueries are computed for each knot. While the dual approach is more compact, it seems to be less suited to handle CQs with answer variables $\vec{x}$ by encodings to languages like disjunctive Datalog. Intuitively, this is because one needs to ground the query to separate Boolean queries, one for each tuple $\vec{c}$ of individuals for $\vec{x}$, to find those $\vec{c}$ where no counterexample to the query can be found.

Finally, knot-shaped mosaics have also been fruitfully applied for CQ answering in restricted DLs, eg. in Horn-$\mathcal{SHIQ}$ [8] where they have been enriched with further structural information.

In the light of the results in this and other papers, it appears that the knot approach is a useful tool to analyze CQ answering in DLs which allows to obtain sharp complexity characterizations, both for general and data complexity.

7.3 Open Issues

Several issues remain for future work. One issue concerns implementation of the approach and optimization of the algorithms, which has not been done yet. Another issue is the application and extension of the knot technique to new DLs, like those in the OWL2 family, as well as a refinement to fragments of $\mathcal{SH}$ and query classes where the algorithm we presented does not yield optimal bounds. This includes, for example, the unrestricted case of $\mathcal{S}$, for which the best bounds currently known are a CONEXPTIME-hardness lower bound and a 2-EXPTIME upper bound [10]. Finally, it would be interesting to extend the current technique to more expressive queries, such as unions of CQs and positive existential queries, or to queries with regular role expressions.

A Appendix: Proof of Proposition 7

Proposition 7. *For each CQ q it holds that* $\mathsf{fd}(\mathsf{fw}(q)) \leq t(\mathsf{fw}(q)) + 1 \leq t(q) + 1$.

Proof. It is easy to verify that $t(\mathsf{fw}(q)) \leq t(q)$: a fork elimination step preserves cycles in $\mathsf{R}^q_*(y)$ for every variable y (but might introduce new ones). Furthermore, it can not increase the number of different predecessors of y; hence, items (3) and (4) of Definition 25 hold for q' if they hold for q. The same holds for the conjunction of (1) and (2).

Let q' result from $\mathsf{fw}(q)$ by removing all atoms $R(x, y)$ where x reaches a cycle in the query graph of $\mathsf{fw}(q)$, i.e., some $z \in \mathsf{R}^{\mathsf{fw}(q)}_+(x)$ exists such that $z \in \mathsf{R}^{\mathsf{fw}(q)}_+(z)$. Note that $\mathsf{fd}(q') = \mathsf{fd}(\mathsf{fw}(q))$. Without loss of generality, we assume that q' contains a single unary atom $C(x)$ for each variable $x \in \mathbf{V}(q')$.

Since q' is acyclic, we can construct q' along a topological sort $x_1 < x_2 < \cdots < x_n$ of its variables, i.e., $R(x_j, x_i) \in q'$ implies $j < i$, starting from x_1 with $C(x_1)$ and adding variable x_i with $C(x_i)$ and all atoms $R(x_j, x_i)$ for $i > 1$.

We show now by induction on $n \geq 1$ that

$$\mathsf{fd}(q') \leq t(q') + 1. \tag{2}$$

Base case. Here $q' = \{C(x_1)\}$ and $\mathsf{fd}(q') = 1$; thus (2) holds for q'.

Induction Step. Suppose we join a variable x_n with $C(x_n)$ and atoms $R_1(x_{n_1}, x_n)$, ..., $R_n(x_{n_{k_n}}, x_n)$ to q' of the assumed form, which yields a query q'' of similar form, and let $A = \{x_{n_1}, \ldots, x_{n_{k_n}}\}$. We consider two cases.

Case 1. Suppose first that $|A| \leq 1$, i.e., x_n is connected to at most one variable in q'. Then, clearly $t(q'') = t(q')$ (condition 4 is violated for the pair x_{n_1}, x_n) and $\mathsf{fd}(q'') = \mathsf{fd}(q')$, which means that (2) holds for q''.

Case 2. Suppose that $|A| = m > 1$, i.e., y is connected to multiple distinct variables $x_{n_1}, \ldots, x_{n_m}$ in q''. In this case,

$$t(q') + (m - 1) \leq t(q'') \tag{3}$$

holds, as each pair x, y in $\mathbf{V}(q')$ that satisfies the conditions 1-4 for $t(q')$ satisfies them for $t(q'')$, and at least $m - 1$ pairs $x_{n,i}, x_n$ satisfy them for $t(q'')$, given that fork elimination is not applicable to any $R(x_{n_i}, x_n)$, $R'(x_{n_j}, x_n)$.

Let $X \subseteq \mathbf{V}(q'')$ be a fork set for q'' such that $|X| = \mathsf{fd}(q'')$. Let $\mathcal{X} = \{X_1, \ldots, X_k\}$ be the set of all maximal $X_i \subseteq X$ (w.r.t. $\subseteq$) that are fork sets for q'. Then for $X_i \neq X_j \in \mathcal{X}$, the sets $\mathsf{R}_*^{q'}(X_i)$ and $\mathsf{R}_*^{q'}(X_j)$ are disjoint and $\bigcup \mathcal{X} = X$. Furthermore, $k \leq m$ must hold: as $X_i \neq X_j \in \mathcal{X}$ must be connected in q'' via y, we have $\mathsf{R}_*^{q''}(X_i) \cap \mathsf{R}_*^{q''}(X_j) = \{y\}$. On the other hand, $\mathsf{R}_*^{q''}(X_i) \cap A \neq \emptyset$ and $\mathsf{R}_*^{q''}(X_j) \cap A \neq \emptyset$. Hence, at most m different X_i exist.

Now consider for the query $q_i' \subseteq q'$ that contains all atoms from q_i on the variables $\mathsf{R}_*^{q'}(X_i)$. Then X_i is a fork set for q_i'; hence by the induction hypothesis for q_i',

$$|X_i| \leq \mathsf{fd}(q_i') \leq t(q_i') + 1.$$

As each pair $x_i, x_j \in \mathbf{V}(q_i')$ satisfies conditions 1-4 for $t(q')$ if it satisfies them for $t(q_i')$, and since the X_i are pairwise disjoint and the queries q_i' are pairwise disconnected, we conclude

$$|X| = \sum_{i=1}^{k} |X_i| \leq \sum_{i=1}^{k} (t(q_i') + 1) \leq t(q') + k \leq t(q') + m.$$

Thus using (3),

$$\mathsf{fd}(q'') = |X| \leq t(q') + m \leq t(q'') + 1,$$

and the induction statement (2) holds for q''; this completes the proof. □

References

[1] F. Baader, C. Lutz, and B. Motik, editors. *Proc. 21st International Workshop on Description Logics (DL2008), Dresden, Germany, May 13-16, 2008*, volume 353 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[2] D. Calvanese, T. Eiter, and M. Ortiz. Answering regular path queries in expressive description logics: An automata-theoretic approach. In *Proc. 22nd Nat. Conf. Artificial Intelligence (AAAI 2007)*, pages 391–396, 2007.

[3] D. Calvanese, T. Eiter, and M. Ortiz. Regular path queries in expressive description logics with nominals. In C. Boutilier, editor, *Proc. 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 714–720. AAAI Press/IJCAI, 2009.

[4] D. Calvanese, E. Franconi, V. Haarslev, D. Lembo, B. Motik, A.-Y. Turhan, and S. Tessaris, editors. *Proc. 2007 International Workshop on Description Logics (DL2007), Brixen-Bressanone, Italy, 8-10 June, 2007*, volume 250 of *CEUR Workshop Proc.* CEUR-WS.org, 2007.

[5] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In P. Doherty, J. Mylopoulos, and C. A. Welty, editors, *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, pages 260–270. AAAI Press, 2006.

[6] D. Calvanese, G. D. Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proc. 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington*, pages 149–158. ACM Press, 1998.

[7] T. Eiter, G. Gottlob, and H. Mannila. Disjunctive datalog. *ACM Transactions on Database Systems*, 22(3):364–417, September 1997.

[8] T. Eiter, G. Gottlob, M. Ortiz, and M. Šimkus. Query answering in the description logic Horn-$\mathcal{SHIQ}$. In S. Hölldobler, C. Lutz, and H. Wansing, editors, *Proc. 11th European Conf. Logics in Artificial Intelligence (JELIA 2008)*, number 5293 in LNCS, pages 166–179. Springer, 2008.

[9] T. Eiter, C. Lutz, M. Ortiz, and M. Šimkus. Query answering in description logics: the knots approach. In H. Ono, M. Kanazawa, and R. de Queiroz, editors, *Proceedings 16th Workshop on Logic, Language, Information and Computation (WoLLIC 2009)*, number 5514 in LNCS, pages 26–36. Springer, 2009.

[10] T. Eiter, C. Lutz, M. Ortiz, and M. Šimkus. Query answering in description logics with transitive roles. In C. Boutilier, editor, *Proc. 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 759–764. AAAI Press/IJCAI, 2009.

[11] T. Eiter, M. Ortiz, and M. Šimkus. Reasoning using knots. In I. Cervesato, H. Veith, and A. Voronkov, editors, *Proc. 15th Int. Conf. Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2008)*, number 5330 in LNCS, pages 377–390. Springer, 2008.

[12] D. Fox and C. P. Gomes, editors. *Proc. 23rd AAAI Conference on Artificial Intelligence (AAAI 2008), Chicago, Illinois, USA, July 13-17, 2008*. AAAI Press, 2008.

[13] B. Glimm, I. Horrocks, C. Lutz, and U. Sattler. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. In M. M. Veloso, editor, *Proc. 20th Int. Joint Conf. Artificial Intelligence (IJCAI 2007)*, pages 399–404, 2007.

[14] B. Glimm, I. Horrocks, and U. Sattler. Conjunctive query entailment for $\mathcal{SHOQ}$. In Calvanese et al. [4].

[15] B. Glimm, I. Horrocks, and U. Sattler. Unions of conjunctive queries in $\mathcal{SHOQ}$. In G. Brewka and J. Lang, editors, *KR*, pages 252–262. AAAI Press, 2008.

[16] B. Glimm, C. Lutz, I. Horrocks, and U. Sattler. Answering conjunctive queries in the $\mathcal{SHIQ}$ description logic. *Journal of Artificial Intelligence Research*, 31:150–197, 2008.

[17] I. Horrocks and S. Tessaris. A conjunctive query language for description logic ABoxes. In *Proc. 17th Nat. Conf. Artificial Intelligence (AAAI 2000)*, pages 399–404, 2000.

[18] U. Hustadt, B. Motik, and U. Sattler. A decomposition rule for decision procedures by resolution-based calculi. In *Proc. 11th Int. Conf. Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2004)*, pages 21–35, 2004.

[19] U. Hustadt, B. Motik, and U. Sattler. Data complexity of reasoning in very expressive description logics. In L. P. Kaelbling and A. Saffiotti, editors, *IJCAI*, pages 466–471. Professional Book Center, 2005.

[20] M. Krötzsch, S. Rudolph, and P. Hitzler. Conjunctive queries for a tractable fragment of OWL 1.1. In K. Aberer et al., editor, *ISWC/ASWC*, volume 4825 of *Lecture Notes in Computer Science*, pages 310–323. Springer, 2007.

[21] A. Y. Levy and M.-C. Rousset. Combining Horn rules and description logics in CARIN. *Artificial Intelligence*, 104(1-2):165–209, 1998.

[22] C. Lutz. Inverse roles make conjunctive queries hard. In Calvanese et al. [4].

[23] C. Lutz. Two upper bounds for conjunctive query answering in $\mathcal{SHIQ}$. In Baader et al. [1].

[24] C. Lutz, U. Sattler, and L. Tendera. The complexity of finite model reasoning in description logics. *Inf. Comput.*, 199(1-2):132–171, 2005.

[25] C. Lutz, F. Wolter, and M. Zakharyaschev. Temporal description logics: A survey. In S. Demri and C. S. Jensen, editors, *TIME*, pages 3–14. IEEE Computer Society, 2008.

[26] M. Marx and Y. Venema. Local variations on a loose theme: Modal logic and decidability. In *Finite Model Theory and Its Applications*, chapter 7, pages 371–429. Springer, June 2007.

[27] I. Németi. Free algebras and decidability in algebraic logic. DSc. thesis, Mathematical Institute of The Hungarian Academy of Sciences, Budapest, 1986.

[28] M. Ortiz, D. Calvanese, and T. Eiter. Characterizing data complexity for conjunctive query answering in expressive description logics. In *AAAI*, pages 275–280. AAAI Press, 2006.

[29] M. Ortiz, D. Calvanese, and T. Eiter. Data complexity of query answering in expressive description logics via tableaux. *J. Automated Reasoning*, 41(1):61–98, 2008.

[30] M. Ortiz, M. Šimkus, and T. Eiter. Conjunctive query answering in $\mathcal{SH}$ using knots. In Baader et al. [1].

[31] M. Ortiz, M. Simkus, and T. Eiter. Worst-case optimal conjunctive query answering for an expressive description logic without inverses. In Fox and Gomes [12], pages 504–510.

[32] V. R. Pratt. Models of program logics. In *FOCS*, pages 115–122. IEEE, 1979.

[33] R. Rosati. On conjunctive query answering in $\mathcal{EL}$. In Calvanese et al. [4].

[34] S. Rudolph, M. Krötzsch, and P. Hitzler. Terminological reasoning in $\mathcal{SHIQ}$ with ordered binary decision diagrams. In Fox and Gomes [12], pages 529–534.

[35] K. Schild. A correspondence theory for terminological logics: Preliminary report. In *IJCAI*, pages 466–471, 1991.

[36] M. Šimkus and T. Eiter. FDNC: Decidable non-monotonic disjunctive logic programs with function symbols. In N. Dershowitz and A. Voronkov, editors, *Proc. 14th Int. Conf. Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2007)*, number 4790 in LNCS, pages 514–530. Springer, 2007. Full paper in *ACM Trans. Computational Logic*, to appear.

[37] S. Tessaris. *Questions and Answers: Reasoning and Querying in Description Logic*. PhD thesis, University of Manchester, Department of Computer Science, 2001.