Query Answering in Description Logics with Transitive Roles*

Thomas Eiter[1], **Carsten Lutz**[2], **Magdalena Ortiz**[1], **Mantas Šimkus**[1]

[1] Institute of Information Systems
Vienna University of Technology, Austria
(eiter|ortiz|simkus)@kr.tuwien.ac.at

[2] Fachbereich Mathematik und Informatik
Universität Bremen, Germany
clu@informatik.uni-bremen.de

Abstract

We study the computational complexity of conjunctive query answering w.r.t. ontologies formulated in fragments of the description logic $\mathcal{SHIQ}$. Our main result is the identification of two new sources of complexity: the combination of transitive roles and role hierarchies which results in 2-ExpTime-hardness, and transitive roles alone which result in co-NExpTime-hardness. These bounds complement the existing result that inverse roles make query answering in $\mathcal{SHIQ}$ 2-ExpTime-hard. We also show that conjunctive query answering with transitive roles, but without inverse roles and role hierarchies, remains in ExpTime if the ABox is tree-shaped.

1 Introduction

One of the main applications of ontologies in computer science is in data access, where an ontology formalizes conceptual information about data that is stored in one or multiple data sources, and this information is used to derive answers when querying the sources. This general setup plays a central role e.g. in ontology-based information integration and in peer-to-peer data management. In all these areas, Description Logics (DLs) and in particular those of the OWL standard by the W3C are popular ontology languages, and conjunctive queries (CQs) are used as a fundamental querying mechanism; cf. [Tessaris, 2001; Glimm *et al.*, 2008b; Ortiz *et al.*, 2008a] and references therein and below.

In spite of prominent applications, studying the computational complexity of answering CQs over OWL ontologies has only recently gained momentum. In particular, it was shown that inverse roles have an impact on complexity: (a) CQ entailment (the decisional variant of CQ answering) over ontologies in the primary OWL fragment $\mathcal{SHIQ}$ is 2-ExpTime-complete [Glimm *et al.*, 2008b; Lutz, 2008], thus harder than standard reasoning tasks such as satisfiability and subsumption which are ExpTime-complete; and (b) the complexity drops to ExpTime-complete if inverse roles are disallowed ($\mathcal{SHIQ}$ is replaced with $\mathcal{SHQ}$) and, additionally, the use of transitive roles in queries is disallowed or seriously restricted, cf. [Lutz, 2008; Ortiz *et al.*, 2008b].

While dropping inverse roles was known to be crucial for obtaining an ExpTime upper bound, the restriction on transitive roles was not. From an application perspective, such a restriction is unsatisfactory as transitive roles play a central role in many ontologies and are used to represent fundamental relations such as "part of" [Sattler, 2000]. However, algorithms for CQ entailment with unrestricted transitive roles in the query are much more intricate than without, see e.g. [Glimm *et al.*, 2008b; Calvanese *et al.*, 2007].

The aim of this paper is to study the computational complexity of CQ entailment in fragments of $\mathcal{SHIQ}$ with no restrictions on transitive roles in queries. Our main contribution is to identify two novel sources of complexity: (1) the combination of transitive roles and role hierarchies and (2) to a lesser degree, transitive roles alone. More precisely, we first show that CQ entailment in $\mathcal{SH}$ ($\mathcal{SHIQ}$ without inverse roles and number restrictions) is 2-ExpTime-hard, and thus 2-ExpTime-complete. Thus, inverse roles are *not* the only reason why CQ entailment in $\mathcal{SHIQ}$ is hard. Interestingly, 2-ExpTime-hardness is hit already with a single role inclusion (or alternatively with a single left identity $r \circ t \sqsubseteq t$) and with an empty ABox (which contains the data). Secondly, we prove that CQ entailment in $\mathcal{S}$ ($\mathcal{SH}$ without role hierarchies) is co-NExpTime-hard. The lower bound applies already to the case where the TBox (conceptual information) is empty.

On the other hand, we show that CQ entailment in $\mathcal{S}$ ontologies where the ABoxes have a tree-shaped relational structure is in ExpTime, and thus ExpTime-complete. This result is interesting for three reasons. Firstly, it is the first ExpTime result for CQ entailment in an expressive DL with unrestricted transitive roles in queries. Secondly, to the best of our knowledge, this is the first case where CQ entailment for tree-shaped ABoxes is easier than the general case: in all existing lower bounds for CQ entailment in fragments of $\mathcal{SHIQ}$, the ABox contains *no* role assertions at all. Thirdly, ExpTime membership may be viewed as an indication that the complexity in the general case is likely to be below 2-ExpTime; a tight upper bound is currently open. Full proofs can be found in [Eiter *et al.*, 2009].

*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).

2 Preliminaries

Knowledge Bases. We assume standard notation for the syntax and semantics of $\mathcal{SH}$ knowledge bases [Glimm *et al.*, 2008b]. In particular, $\mathsf{N_C}$, $\mathsf{N_R}$, and $\mathsf{N_I}$ are countably infinite and disjoint sets of *concept names*, *role names*, and *individual names*. *Concepts* are inductively defined: (a) each $A \in \mathsf{N_C}$ is a concept, and (b) if C, D are concepts and $r \in \mathsf{N_R}$ is a role, then $C \sqcap D$, $C \sqcup D$, $\neg C$, $\forall r.C$ and $\exists r.C$ are concepts. A *TBox* is a set of concept inclusions $C \sqsubseteq D$, role inclusions $r \sqsubseteq s$, and transitivity statements $\mathsf{trans}(r)$. An *ABox* is a set of *assertions* $C(a)$ and $r(a, b)$. A *knowledge base (KB)* is a pair $(\mathcal{T}, \mathcal{A})$ consisting of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. We use $\mathcal{I}$ to denote an interpretation, $\Delta^{\mathcal{I}}$ for its domain, and $C^{\mathcal{I}}$ and $r^{\mathcal{I}}$ for the interpretation of a concept C and of a role r, respectively. We denote by $\mathsf{Ind}(\mathcal{A})$ the set of all individual names in an ABox $\mathcal{A}$. $\mathcal{S}$ is the fragment of $\mathcal{SH}$ that disallows role inclusions.

Conjunctive Query Answering. Let $\mathsf{N_V}$ be a countably infinite set of *variables*. A *conjunctive query* (CQ) over a KB $\mathcal{K}$ is a finite set of atoms of the form $A(v)$ or $r(v, v')$, where $v, v' \in \mathsf{N_V}$, A is a concept name and r is a role, both occurring in $\mathcal{K}$.[1] For a CQ q over $\mathcal{K}$, let $\mathsf{Var}(q)$ denote the variables occurring in q. A *match for q in an interpretation $\mathcal{I}$* is a mapping $\pi : \mathsf{Var}(q) \to \Delta^{\mathcal{I}}$ such that (i) $\pi(v) \in A^{\mathcal{I}}$ for each $A(v) \in q$, and (ii) $(\pi(v), \pi(v')) \in r^{\mathcal{I}}$ for each $r(v, v') \in q$. We write $\mathcal{I} \models q$ if there is a match for q in $\mathcal{I}$. If $\mathcal{I} \models q$ for every model $\mathcal{I}$ of $\mathcal{K}$, then $\mathcal{K}$ *entails* q, written $\mathcal{K} \models q$. The *query entailment problem* is to decide, given $\mathcal{K}$ and q, whether $\mathcal{K} \models q$.

Forest Models. When studying CQ answering in the $\mathcal{SHIQ}$ family of DLs, it suffices to consider models that have a forest-like shape; informally, such a model $\mathcal{I}$ consists of two parts: an ABox part that consists of the interpretations $a^{\mathcal{I}}$ of the individuals a in $\mathcal{K}$ and has an unrestricted relational structure, and a forest part that is a collection of trees whose roots are elements of the ABox part and that are otherwise disjoint from the ABox part. The following result can be found e.g., in [Glimm *et al.*, 2008b].

Proposition 1. *For every $\mathcal{SH}$-knowledge base and CQ q, if $\mathcal{K} \not\models q$, then $\mathcal{K}$ has a forest model $\mathcal{I}$ such that $\mathcal{I} \not\models q$.*

3 Query Answering in $\mathcal{SH}$

It follows from a number of existing results that CQ entailment in $\mathcal{SH}$ is in 2-ExpTime [Calvanese *et al.*, 2007; Glimm *et al.*, 2008a; Ortiz *et al.*, 2008b]. We provide a matching lower bound.

Theorem 1. *CQ entailment in $\mathcal{SH}$ is 2-*ExpTime*-complete.*

It is well-known that there is an exponentially space bounded *Alternating Turing Machine (ATM)* $\mathcal{M}$ whose word problem is 2-ExpTime-hard [Chandra *et al.*, 1981]. To prove the lower bound from Theorem 2, we reduce this word problem.

Recall that the state set Q of an ATM is partitioned into *existential* ($Q_\exists$) and *universal* ($Q_\forall$) states. An ATM with only existential states can be viewed as a standard nondeterministic TM, which accepts a word iff there exists a sequence of successive configurations that starts in the *initial configuration*, with initial state q_0 and the input word w on the tape, and ends in an accepting state q_{acc}. For general ATMs, these sequences become *trees* of configurations, where branching is caused by universal states (there is a successive configuration for each transition in $\delta(q, a)$ with $q \in Q_\forall$). Such a *computation tree* is *accepting* if q_{acc} is reached on all paths. For details, see [Chandra *et al.*, 1981].

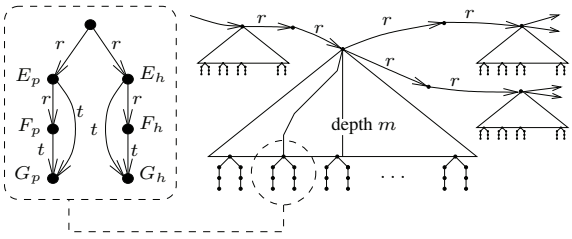

Figure 1: The structure of models.

For each input w to $\mathcal{M}$, we define a KB $\mathcal{K}_w$ and a query q_w such that $\mathcal{M}$ accepts w iff $\mathcal{K}_w \not\models q_w$. In fact, forest models $\mathcal{I}$ of $\mathcal{K}_w$ with $\mathcal{I} \not\models q_w$ will represent an accepting computation of $\mathcal{M}$ on w. More precisely, $\mathcal{I}$ is an accepting computation tree each of whose nodes is the root of a *configuration tree*. The latter are binary trees of depth $m := |w|$ (length of w) that represent configurations using their 2^m leaves to store the tape contents. This is illustrated in Figure 1; the initial configuration tree is existential and thus has a single successor configuration tree. This (magnified) successor is universal and has two successor configuration trees.

To enforce this structure, we need some technical tricks. In particular, each configuration tree will represent *two* configurations: the *current configuration* K_h and the *previous* configuration K_p. We use $\mathcal{K}_w$ to ensure locally at each configuration tree that K_h is indeed a successor configuration of K_p. The query q_w is then used to globally guarantee that the K_p value of each configuration tree is identical to the K_h value of the predecessor in the computation tree. We will call a computation tree *proper* if it satisfies the latter condition.

We now give a precise definition of how configuration trees and computation trees are represented as a model. A single, non-transitive role r is used for the edges of computation trees and of configuration trees. Observe that, as shown in Figure 1, we use *two* r-edges between two consecutive configuration trees. We also use a transitive role t, to be explained later. The alphabet symbols Σ of $\mathcal{M}$ and the states Q are used as concept names. We also use the concept names from $\mathbf{B} := \{B_1, \ldots, B_m\}$ to encode addresses of tape cells in binary. For a node n of a forest model $\mathcal{I}$ and $i < 2^m$, we write $\mathsf{adr}^{\mathcal{I}}(n) = i$ if the truth values of $B_1^{\mathcal{I}}, \ldots, B_m^{\mathcal{I}}$ at n encode the number i. A tape cell with address i and content $a \in \Sigma$ is represented by a node n with $\mathsf{adr}^{\mathcal{I}}(n) = i$ that satisfies the concept name a. If the head is currently on the cell and $\mathcal{M}$'s state is q, then n also satisfies q; otherwise, n satisfies the concept name nil.

To later on ensure properness using the query, we use additional nodes and concept names. The latter are E_h, E_p, F_h, F_p, G_h, and G_p, used as markers; and the concept names from $\mathbf{Z} := \{Z_{a,q} \mid a \in \Gamma \text{ and } q \in Q \cup \{nil\}\}$. The additional nodes are attached to the leaves of configuration trees,

[1] Individuals in q can be simulated and queries with answer variables can be reduced to the considered Boolean CQs as usual.

as indicated on the left-hand side of Figure 1 and detailed in the subsequent definition. Intuitively, nodes labeled E_h store the current configuration and nodes labeled E_p the previous.

Definition 1 (i-cell). Let $\mathcal{I}$ be an interpretation and $i < 2^m$. We call $n \in \Delta^{\mathcal{I}}$ an *i-cell* if the following hold:

(a) n has r-successors n_p and n_h that respectively satisfy E_p and E_h, with $\mathsf{adr}^{\mathcal{I}}(n_p) = \mathsf{adr}^{\mathcal{I}}(n_h) = i$, and such that both satisfy exactly one $a \in \Sigma$ and exactly one $q \in Q \cup \{nil\}$.

(b) n_p (resp., n_h) has an r-successor n'_p (resp., n'_h) that satisfies F_p (resp., F_h), with $\mathsf{adr}^{\mathcal{I}}(n'_p) = \mathsf{adr}^{\mathcal{I}}(n'_h)$ the bit-wise complement of i, and such that for all $a \in \Sigma$, $q \in Q \cup \{nil\}$:

(i) n_h satisfies $Z_{a,q}$ iff n_h does not satisfy both a and q;

(ii) n'_p satisfies $Z_{a,q}$ iff n_p does not satisfy both a and q;

(iii) n'_h and n_p satisfy $Z_{a,q}$;

(c) n'_p (resp., n'_h) has a t-successor n''_p satisfying G_p (resp., n''_h satisfying G_h) such that n''_p (resp., n''_h) is also a t-successor of n_p (resp., n_h).

We simply speak of a cell if i is unimportant. Note that the ability of $\mathcal{SH}$ to express (c) in Definition 1 via the axioms $r \sqsubseteq t$ and $\mathsf{trans}(t)$ is crucial for the reduction. The same condition can be expressed via a so-called *left identity* $r \circ t \sqsubseteq t$.

We now define (q, a, i)-configuration nodes, which are the roots of configuration trees, and (models that encode) computation trees. A node n' is an r^m-successor of a node n, if n' is reachable from n by traveling m r-edges.

Definition 2 ((q, a, i)-configuration node, Computation tree). Let $\mathcal{I}$ be an interpretation. We call $n \in \Delta^{\mathcal{I}}$ a *(q, a, i)-configuration node* if (1) it has an s^m-successor that is a j-cell, for each $j < 2^m$ and (2) the E_h-node of the i-cell satisfies q and a, and all other j-cells have nil in their E_h-nodes.

We call $\mathcal{I}$ a *computation tree* for w if $\mathcal{I}$ is tree-shaped and

1. the root ϵ of $\mathcal{I}$ has an r-successor n that is a $(q_0, a, 0)$-configuration node whose i-cells describe the initial configuration for input w;

2. for each (q, a, p)-configuration node n, if $q \in Q_\exists$ (resp., $q \in Q_\forall$), then for some (resp., for each) tuple $(q', a', M) \in \delta(q, a)$ we have:

(i) there exists an r^2-successor node n' that is an (q', a'', p')-configuration node with $p' = p + M$, where $M \in \{-1, +1\}$ is the executed move,

(ii) the E_h node of the p-cell of n' satisfies a', and,

(iii) for all j-cells c of n' where $j \neq p$, if the E_p node of c satisfies $a \in \Sigma$, then the E_h node of c also satisfies a.

We call $\mathcal{I}$ *accepting*, if $q = q_{\mathsf{acc}}$ in each (q, a, i)-configuration for which there are no successor configurations. $\mathcal{I}$ is *proper*, if for each pair of successive configuration nodes n_1, n_2 and each $i < 2^m$, the i-cell of n_1 has the same (q, a)-label in its E_h-node as the i-cell of n_2 in its E_p-node.

It is not hard to verify that there is a one-to-one correspondence between accepting proper computation trees for w and accepting computations of $\mathcal{M}$ on w.

Proposition 2. *$\mathcal{M}$ accepts w iff there exists an accepting proper computation tree for w.*

It is not too difficult to show the following.

Proposition 3. *Given w, we can build in polynomial time a KB $\mathcal{K}_w$ whose forest models are exactly the accepting computation trees for w.*

As already mentioned, we use the query q_w to test whether the tree is proper. More precisely, q_w should have a match in a computation tree iff that tree is *not* proper. We start with a characterization of (im)properness in terms of the auxiliary concept names from above. In the following, we say that two cells n and n' are *A-conspicuous*, with A a concept name, if

(†) A is true at the E_h-node of n and the E_p-node of n', or

(‡) A is true at the F_h-node of n and the F_p-node of n'.

Proposition 4. *A computation tree $\mathcal{I}$ is not proper iff ($\star$) there exists a cell n in some configuration K and a cell n' in a successor configuration of K such that for all $A \in \mathbf{B} \cup \mathbf{Z}$, n and n' are A-conspicuous.*

The above proposition holds due to the way auxiliary labels are defined. First note that if n, n' are cells of two successive configurations in $\mathcal{I}$, then the conditions imposed on $\mathsf{adr}^{\mathcal{I}}(\cdot)$ in Definition 1 imply that $\mathsf{adr}^{\mathcal{I}}(n) = \mathsf{adr}^{\mathcal{I}}(n')$ iff for all $A \in \mathbf{B}$, n and n' are A-conspicuous; this is because bit-wise complement is used for the addresses of F_p- and F_h-nodes.

Now suppose that $\mathcal{I}$ is proper and let n, n' be cells of two successive configurations. If they are not A-conspicuous for some $A \in \mathbf{B}$ then, as required, ($\star$) is violated. If there is no such $A \in \mathbf{B}$, then $\mathsf{adr}^{\mathcal{I}}(n) = \mathsf{adr}^{\mathcal{I}}(n')$. As $\mathcal{I}$ is proper, the E_h-node of n and the E_p-node of n' satisfy the same $q \in Q$ and $a \in \Sigma$. By (b.i), $Z_{a,q}$ is false at the E_h-node of n; by (b.ii), $Z_{a,q}$ is false at the F_p-node of n'. Hence, n, n' are not $Z_{a,q}$-conspicuous and ($\star$) is violated. Conversely, let $\mathcal{I}$ be improper. Then there exist two j-cells n and n' of two successive configurations such that the E_h-node of n and the E_p-node of n' satisfy different pairs (q, a) and (q', a'). As $\mathsf{adr}^{\mathcal{I}}(n) = \mathsf{adr}^{\mathcal{I}}(n')$, n and n' are A-conspicuous for all $A \in \mathbf{B}$. By (b.iii), $Z_{q,a}$ is true at the F_h-node of n; by (b.ii) and since $(q, a) \neq (q', a')$, $Z_{q,a}$ is also true at the F_p-node of n'. We can argue symmetrically that $z_{a',q'}$ is true at the E_h-node of n and the E_p-node of n'. For $(q'', a'') \notin \{(q, a), (q', a')\}$, $Z_{a'',q''}$ holds at the E_h-, E_p-, F_h-, and F_p-nodes of both n and n'. In summary, n and n' are A-conspicuous for all $A \in \mathbf{Z}$ and thus ($\star$) is satisfied.

It thus remains to find a query q_w that has a match iff ($\star$) is satisfied. The structure of q_w is displayed in Figure 2(II).

We obtain q_w by taking, for each $A \in \mathbf{B} \cup \mathbf{Z}$, a copy of the basic query $q(A, u, v)$ in Figure 2(I) such that the different copies share only the variables u and v, and then taking the union. Intuitively, $q(A, u, v)$ deals with A-conspicuousness, and the shared variables u, v ensure that the different component queries speak about the same cells n, n'. In more detail, let n, n' be cells of two successive configurations that are A-conspicuous for all $A \in \mathbf{B} \cup \mathbf{Z}$. We can find a match for q_w as follows: start with matching u on the G_h-node of n and v on the G_p-node of n'. Now take an $A \in \mathbf{B} \cup \mathbf{Z}$. If (†) applies, then match y^A_{m+1} on the E_h-node of n and z^A_{m+1} on the E_p-node of n'; if (‡) applies, then match y^A_{m+1} on the F_h-node of n and z^A_{m+1} on the F_p-node of n'. The matches of all other

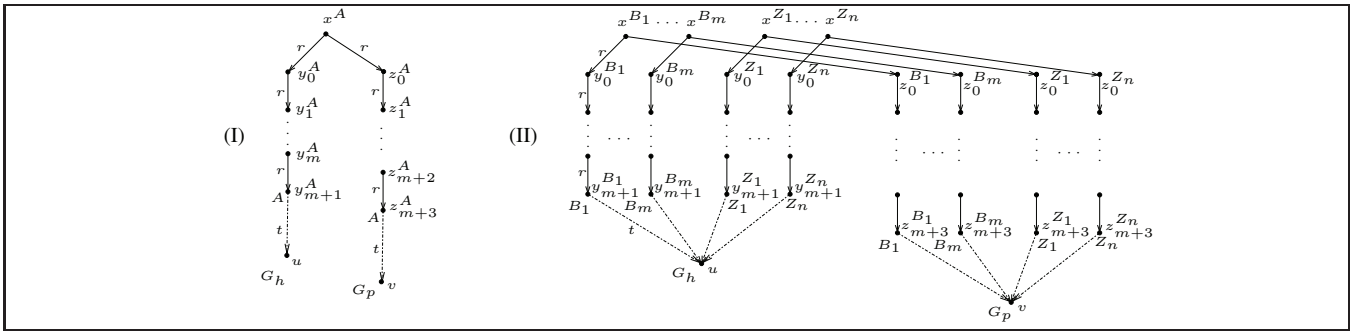

Figure 2: The basic query $q(A, u, v)$ and the final query q_w.

variables are now uniquely determined by the (non-transitive) role edges in the query. In particular, the lengths of the role chains in the query ensure that x^A will be matched to the root of n in case (‡) and to the root's predecessor in case (†). Observe that the paths labeled with z-variables are exactly two steps longer than those labeled with y-variables, and thus the query only relates n and n' if they belong to successor configurations. In summary, it is possible to show that

Proposition 5. *A computation tree $\mathcal{I}$ is proper iff $\mathcal{I} \not\models q_w$.*

Together with Propositions 2 and 3, this yields the desired reduction, establishing the lower bound from Theorem 2.

4 Query Answering in $\mathcal{S}$

The reduction in the previous section crucially exploits the presence of role inclusions and transitive roles. In particular, the structure shown on the left-hand side of Figure 1 cannot be enforced if either of these expressive means is dropped. Role hierarchies alone do not suffice to make CQ entailment harder than satisfiability checking and other EXPTIME-complete standard reasoning tasks: it is known that CQ entailment in $\mathcal{ALCH}$, which is $\mathcal{SH}$ without transitive roles, is only EXPTIME-complete [Lutz, 2008; Ortiz *et al.*, 2008b]. In contrast and as we show next, transitive roles alone suffice to make CQ answering harder than standard reasoning.

Theorem 2. *CQ entailment in $\mathcal{S}$ is* CO-NEXPTIME*-hard even without TBoxes and with acyclic ABoxes.*

This result is shown by a reduction to CQ non-entailment from a NEXPTIME-complete variant of the tiling problem where the task is to tile a $2^n \times 2^n$-torus. The reduction uses only a single transitive role and no other role, no TBox, and an ABox whose relational structure is a directed acyclic graph, but not a tree. Because of space limitation, we refer to [Eiter *et al.*, 2009] for a detailed construction and give only a rough idea why transitive roles make CQ entailment hard. Viewed on a high level of abstraction, most algorithms for CQ entailment exploit forest models and work by splitting the problem into a number of subproblems: one for each individual a in the input ABox $\mathcal{A}$. This splitting also involves splitting the input query q into subqueries: a match of q in a forest model $\mathcal{I}$ may send some variables to the subtree below a and some variables to other parts of $\mathcal{I}$, and we obtain a subquery by including only variables of the former kind. Now, the crucial observation is as follows: without transitive roles, only polynomially many subqueries are generated for each a whereas exponentially many subqueries have to be considered when transitive roles are admitted.

As we show next, the described effect vanishes in the case of tree-shaped ABoxes. This case is relevant, e.g., when the ABox is obtained by translating an XML document. An ABox $\mathcal{A}$ is *tree-shaped* if the directed graph with nodes $\mathsf{Ind}(\mathcal{A})$ and edges $\{(a, b) \mid r(a, b) \in \mathcal{A}\}$ is a tree and $r(a, b), r'(a, b) \in \mathcal{A}$ imply $r = r'$. We aim to show

Theorem 3. *CQ entailment in $\mathcal{S}$ is* EXPTIME*-complete if ABoxes are tree-shaped.*

It is well-known that CQ entailment in $\mathcal{S}$ is EXPTIME-hard even with empty ABoxes and thus it remains to show the upper bound. We start with a simple observation.

Proposition 6. *For a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, where $\mathcal{A}$ is tree-shaped, we can build in polynomial time a KB $\mathcal{K}' = (\mathcal{T}, \{C_{\mathcal{A}}(a)\})$ such that $\mathcal{K} \models q$ iff $\mathcal{K}' \models q$ for every CQ q.*

It thus suffices to give an EXPTIME algorithm for CQ entailment in $\mathcal{S}$ with ABoxes of the form $\{C_0(a)\}$. From now on, let $\mathcal{K} = (\mathcal{T}, \{C_0(a)\})$ be a KB and q a CQ for which we decide $\mathcal{K} \models q$.

We set $\mathsf{Tr}(\mathcal{K}) := \{r \in \mathsf{N_R} \mid \mathsf{trans}(r) \in \mathcal{T}\}$. We assume w.l.o.g. that C_0 is in *negation normal form* (NNF), i.e. negation is only applied to concept names, and that $\mathcal{T}$ has the form $\{\top \sqsubseteq C_{\mathcal{T}}\}$ with $C_{\mathcal{T}}$ in NNF. We may also assume w.l.o.g. that CQs are connected (a disconnected query can be answered by separately posing each connected subquery).

We can limit our attention to certain canonical models and a certain kind of query that we call a *pseudo-tree query*.

Definition 3 (Canonical Model). We use $\mathsf{sub}(\mathcal{K})$ to denote the set of all subconcepts of concepts occurring in $\mathcal{K}$. A *canonical model* for $\mathcal{K}$ is a model $\mathcal{I}$ of $\mathcal{A}$ such that (i) $\mathcal{I}$ satisfies all concept inclusions in $\mathcal{T}$ (but not necessarily the transitivity axioms); (ii) $(\Delta^{\mathcal{I}}, \bigcup_{r \in \mathsf{N_R}} r^{\mathcal{I}})$ is a tree with root $a^{\mathcal{I}}$ and whose out-degree is bounded by the cardinality of $\mathsf{sub}(\mathcal{K})$; (iii) $r^{\mathcal{I}} \cap s^{\mathcal{I}} = \emptyset$ whenever $r \neq s$; (iv) for all $\forall t.C \in \mathsf{sub}(\mathcal{K})$ with $t \in \mathsf{Tr}(\mathcal{K})$ and all $(d, e) \in t^{\mathcal{I}}$, $d \in (\forall t.C)^{\mathcal{I}}$ implies $e \in (\forall t.C)^{\mathcal{I}}$.

Due to the non-transitivity of transitive roles in canonical models $\mathcal{I}$, we have to work with a relaxed version of a match that becomes a match when, for every $r \in \mathsf{Tr}(\mathcal{K})$, $r^{\mathcal{I}}$ is replaced with its transitive closure.

Definition 4 (Pre-match). Let $\mathcal{I}$ be a canonical model of $\mathcal{K}$. We call $\pi : \mathsf{Vars}(q) \to \Delta^{\mathcal{I}}$ a *pre-match* for q in $\mathcal{I}$ if (a)

$\pi(u) \in A^{\mathcal{I}}$ for each $A(u) \in q$, (b) $(\pi(u), \pi(v)) \in r^{\mathcal{I}}$ for each $r(u,v) \in q$ with $r \notin \mathsf{Tr}(\mathcal{K})$, and (c) for each $t(u,v) \in q$ with $t \in \mathsf{Tr}(\mathcal{K})$, there is a sequence $d_0, \ldots, d_n \in \Delta^{\mathcal{I}}$ such that $d_0 = \pi(u)$, $d_n = \pi(v)$ and $(d_i, d_{i+1}) \in t^{\mathcal{I}}$ for all $i < n$. We write $\mathcal{I} \models^{\mathsf{pre}} q$, if there is a pre-match for q in $\mathcal{I}$.

We now define pseudo-tree queries.

Definition 5 (Role Cluster, Pseudo-tree Query). Let q be a CQ. For each $t \in \mathsf{Tr}(\mathcal{K})$, $\sim_t$ denotes the smallest equivalence relation over $\mathsf{Var}(q)$ such that $t(v,v') \in q$ implies $v \sim_t v'$. An equivalence class c_t of $\sim_t$ is called a *(transitive) cluster* of q. For each non-transitive role s, a *(non-transitive) cluster* of q is a set $c_s = \{u, v\}$ with $s(u,v) \in q$. Now, a connected CQ q is a *pseudo-tree query* if it satisfies:

(a) if c_r is a cluster of q and $s(u,v), s'(u',v') \in q$ with $v, v' \in c_r$ and $s, s' \neq r$, then $s = s'$, $u = u'$, $v = v'$;

(b) q is acyclic, i.e., it does not contain atoms $r_0(v_0, v_1)$, $\ldots, r_n(v_n, v_{n+1})$ with $v_{n+1} = v_0$.

A cluster c_r of q is *initial* if no $v \in c_r$ has an incoming edge $r'(v', v) \in q$ with $r \neq r'$.

Intuitively, a pseudo-tree can be viewed as a tree of clusters with an additional root; the root is a predecessor of every initial cluster (there can be more than one) and there is an edge between two clusters if they share an element. It is easy to see that clusters cannot share more than one element. Each transitive cluster in a pseudo-tree query describes a subquery that is an acyclic directed graph.

Definition 6. Let q, q' be conjunctive queries. Then q' is *obtained from q by fork elimination*, if q' results from q by one of the following operations:

- select $r(u,v), r(u',v) \in q$ with $u \neq u'$ and $r \notin \mathsf{Tr}(\mathcal{K})$, and identify u and u';
- select $r(u,v), r(u',v') \in q$ with $v \neq v'$ and v, v' in a cluster c_s of q with $s \neq r$, and identify v and v'.

We say that q' is a *maximal fork rewriting* of q if q' is obtained from q by exhaustive fork elimination.

It can be shown that the maximal fork rewriting is unique and computable in polynomial time. Moreover, it can be checked in polynomial time whether a query is a pseudo-tree query. Hence, the following proposition allows us to restrict our attention to canonical models and pseudo-tree queries.

Proposition 7. *Let q be a CQ, and let q' be the maximal fork rewriting of q. Then $\mathcal{K} \not\models q$ iff (i) q' is not a pseudo-tree query, or (ii) $\mathcal{I} \not\models^{\mathsf{pre}} q'$ for some canonical model $\mathcal{I}$ of $\mathcal{K}$.*

In what follows, assume that the input query q is a pseudo-tree query. We want to decide whether there is a canonical model $\mathcal{I}$ of $\mathcal{K}$ such that $\mathcal{I} \not\models^{\mathsf{pre}} q$. Our approach is based on *knots* [Ortiz *et al.*, 2008c].

Definition 7 (Knot). A *$\mathcal{T}$-type* is a set $\tau \subseteq \mathsf{sub}(\mathcal{T})$ that satisfies, for all $C, D \in \mathsf{sub}(\mathcal{T})$: (a) $C \in \tau$ implies $\neg C \notin \tau$, (b) if $C \sqcap D \in \tau$, then $\{C, D\} \subseteq \tau$, (c) if $C \sqcup D \in \tau$, then $C \in \tau$ or $D \in \tau$, and (d) $C_{\mathcal{T}} \in \tau$. A *knot for $\mathcal{T}$* is a pair $\kappa = (\tau, S)$ with τ a $\mathcal{T}$-type and S a set of pairs (r, τ') such that r is a role name that occurs in $\mathcal{T}$, τ' is a $\mathcal{T}$-type, and in addition:

(1) if $\exists r.C \in \tau$, then $C \in \tau'$ for some $(r, \tau') \in S$;

(2) if $\forall r.C \in \tau$, then $C \in \tau'$ for all $(r, \tau') \in S$;

(3) if $\forall r.C \in \tau \wedge r \in \mathsf{Tr}(\mathcal{K})$, then $\forall r.C \in \tau'$ for all $(r, \tau') \in S$;

(4) $|S| \leq |\mathsf{sub}(\mathcal{K})|$.

A knot $\kappa = (\tau, S)$ can be viewed as describing a fragment of a canonical model that consists of a node which satisfies the concepts in τ and its successors, as described by S. Our algorithm will represent canonical models as a set of knots. In fact, it is not hard to come up with conditions which guarantee that a given set of knots can be assembled into a canonical model. The difficulty is to ensure that, in the represented canonical model, there is no pre-match of q.

The general idea to overcome this difficulty is as follows. Consider a top-down walk though a (tree-shaped!) canonical model $\mathcal{I}$. To avoid a pre-match of the pseudo-tree query q, we 'track' q through the tree, switching to a subquery of q whenever we are able to match at least one variable. For example, if $q = \{A(v_0), B(v_0'), r(v_0, v_1), r(v_0', v_1)\}$ with r transitive and we are currently considering an element $d \in \Delta^{\mathcal{I}}$ such that $d \in A^{\mathcal{I}} \setminus B^{\mathcal{I}}$, then we switch from q to $q' := \{B(v_0'), r(v_0', v_1)\}$ because v_0 (but not v_0') can be matched to d; we then track q' starting from all r-successors of d in $\mathcal{I}$; if e is such a successor with $e \notin B^{\mathcal{I}}$, condition (c) of pre-matches forces us to continue to track q' (without any modifications since v_0' cannot be matched at e) at the r-successors of e; and so on. When we are left with a query that consists of only one variable, we ensure that this variable matches *nowhere*. In summary, we thus use an *eager matching* approach (matching variables as early as possible) to ensure that q matches nowhere in $\mathcal{I}$. This general idea is complicated by the mixture of transitive and non-transitive roles in the query, and by the fact that we have to implement it in terms of knots rather than directly in terms of models.

To track subqueries of q, we need a means to identify such subqueries.

Definition 8 (Query Pointer). A *query pointer* for q is a pair (c_r, V) with c_r a cluster of q and $V \subseteq c_r$ nonempty. If c_r is a cluster of q, $v \in c_r$, and there is an $s(v,v') \in q$ with $s \neq r$, then the query pointer (c_s, c_s) *is possible from* v, where c_s is the (unique!) s-cluster in q with $v, v' \in c_s$.

Intuitively, a query pointer $P = (c_r, V)$ represents the subquery of q generated by the variables that are reachable from the variables in V. By definition of clusters, there are two types of query pointers (c_r, V): (i) r non-transitive and $|V| \in \{1, 2\}$; and (ii) r transitive and V of any cardinality ≥ 1.

Clearly, a pre-match of a query can concern more than a single knot. To implement the tracking of queries based on knots, we thus extend knots with bookkeeping information.

Definition 9 (Marked Knot). A *q-marking* for q is a set Γ of query pointers such that for all clusters c_r of q, there is at most one pointer $(c_r, V) \in \Gamma$. A *marked knot* is a pair (κ, ν), where $\kappa = (\tau, S)$ is a knot and ν assigns a q-marking to each element in $S \cup \{\varepsilon\}$.

As a convention, $\nu(\varepsilon)$ is the marking of the root of κ. Intuitively, $(c_r, V) \in \nu(\varepsilon)$ means that we are currently tracking the subquery of q identified by (c_r, V) at the element of the model that is identified by the root node of the knot, and similarly for $(c_r, V) \in \nu(s, \tau)$.

procedure Knot-Elim($\mathcal{K}, q$) with $\mathcal{K} = (\mathcal{T}, \{C_0(a)\})$
  Compute the set $\mathfrak{K}_0$ of marked q-avoiding knots
  $i := 0$
  repeat
    $i := i + 1$
    $\mathfrak{K}_i := \mathfrak{K}_{i-1} \setminus \{(\kappa, \nu) \in \mathfrak{K}_{i-1} \mid (\kappa, \nu)$ bad in $\mathfrak{K}_{i-1}\}$
  until $\mathfrak{K}_i = \mathfrak{K}_{i-1}$
  if there is a $(\kappa, \nu) \in \mathfrak{K}_i$ with $\kappa = (\tau, S)$ s.t. $C_0 \in \tau$ then
    return "$\mathcal{K}$ does not entails q"
  else return "$\mathcal{K}$ entails q"

Figure 3: The knot elimination algorithm.

To ensure that the query is properly tracked, we propagate markings from the root of a knot to its successors. For $V \subseteq \mathsf{Vars}(q)$, we use $\mathsf{min}(V) \subseteq \mathsf{Vars}(q)$ to denote the set of those $v \in V$ for which there exists no $r(v', v) \in q$ with $v' \in V$.

Definition 10 (q-avoiding). A marked knot (κ, ν) with $\kappa = (\tau, S)$ is *q-avoiding* if (I) $\nu(\varepsilon)$ contains a query pointer (c_r, V) with c_r initial; and (II) the following conditions hold for each $(c_r, V) \in \nu(\varepsilon)$:

1. If $r \notin \mathsf{Tr}(\mathcal{K})$, $V = \{v, v'\}$, and $r(v, v') \in q$, then either
   (a) for some A, we have $A(v) \in q$ and $A \notin \tau$, or
   (b) $(c_r, \{v'\}) \in \nu(r, \tau')$ for all $(r, \tau') \in S$;
2. If $r \notin \mathsf{Tr}(\mathcal{K})$ and $V = \{v\}$, then either
   (a) for some A, we have $A(v) \in q$ and $A \notin \tau$, or
   (b) $P' \in \nu(\varepsilon)$ for a query pointer P' possible from v;
3. If $r \in \mathsf{Tr}(\mathcal{K})$, then there is some set $M \subseteq \mathsf{min}(V)$ s.t.:
   (a) for each $v \in \mathsf{min}(V) \setminus M$, either (i) for some A, we have $A(v) \in q$ and $A \notin \tau$, or (ii) $P' \in \nu(\varepsilon)$ for some query pointer P' possible from v, and
   (b) there is a non-empty $V' \subseteq V \setminus M$ with $(c_r, V') \in \nu(r, \tau')$ for all $(r, \tau') \in S$.

Intuitively, Definition 10 implements the tracking of q described above. Conditions (1.b), (2.b), and (3.b) push the query to successor nodes. In Condition (3), M is the set of variables to be matched at the 'current' node, and (3.a) implements eager matching. The final avoidance of the query is implemented via Conditions (2) and (3) when $|V| = 1$. Condition (I) is needed to re-initiate the tracking process, e.g., when we have traveled a role that does not occur in the query.

Our algorithm is of the type elimination kind introduced in [Pratt, 1979], but works on marked knots instead of on types. It rests on the following definition.

Definition 11 (Bad). Let $\mathfrak{K}$ be a set of marked knots and $(\kappa, \nu) \in \mathfrak{K}$ with $\kappa = (\tau, S)$. We say that (κ, ν) is *bad* in $\mathfrak{K}$, if there is some $(r, \tau') \in S$ for which there is no $(\kappa_s, \nu_s) \in \mathfrak{K}$ with $\kappa_s = (\tau_s, S_s)$ such that $\tau_s = \tau'$ and $\nu_s(\varepsilon) = \nu(r, \tau')$.

The algorithm is given in Figure 3. Soundness and completeness are established in [Eiter *et al.*, 2009].

Proposition 8. *The Knot-Elim algorithm is sound, complete, and terminates.*

To establish Theorem 3, it remains to show that Knot-Elim runs in exponential time. For this, it clearly suffices to show that the number of marked knots is only exponential in the size of $\mathcal{K}$ and q. Let n be the size of $\mathcal{K}$ and m the size of q. The number of $\mathcal{T}$-types is bounded by 2^n and the number of knots by $2^{\mathcal{O}(n^2)}$ (note cond. 4 in Def.7); the number of query pointers is bounded by $2^{\mathcal{O}(m)}$ and the number of q-markings by $2^{\mathcal{O}(m^2)}$. It follows that there are $2^{\mathcal{O}(n^2 m^2)}$ marked knots.

5 Related Work and Conclusions

We have shown that CQ entailment in the DL $\mathcal{SH}$, which supports transitive roles and role hierarchies, is 2-ExpTime-hard, and thus provably harder than standard reasoning tasks such as satisfiability and instance checking, which are ExpTime-complete. We have also shown that the problem is in ExpTime for $\mathcal{S}$ when ABoxes are tree-shaped, but co-NExpTime-hard in general. A tight bound remains open, but we consider it likely that CQ entailment in $\mathcal{S}$ is simpler than 2-ExpTime, with co-NExpTime being a good candidate.

The 2-ExpTime hardness for $\mathcal{SH}$ and for $\mathcal{ALCI}$ [Lutz, 2008] matches known upper bounds for *unions of CQs* in $\mathcal{SHIQ}$ [Glimm *et al.*, 2008b] and for the even more expressive *two-way positive regular path queries* in $\mathcal{ALCQI}b_{reg}$ [Calvanese *et al.*, 2007]. This shows that once either inverse roles or role hierarchies and transitivity are allowed, both the query language and the DL can be significantly extended without further increase of the worst case complexity.

References

[Calvanese *et al.*, 2007] D. Calvanese, T. Eiter, and M. Ortiz. Answering regular path queries in expressive description logics: An automata-theoretic approach. In *Proc. AAAI 2007*, pp. 391–396.

[Chandra *et al.*, 1981] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[Eiter *et al.*, 2009] T. Eiter, C. Lutz, M. Ortiz, and M. Šimkus. Query answering in description logics with transitive roles. INFSYS RR-1843-09-02. TU Vienna, 2009.

[Glimm *et al.*, 2008a] B. Glimm, I. Horrocks, and U. Sattler. Unions of conjunctive queries in $\mathcal{SHOQ}$. In *Proc. KR 2008*, pp. 252–262, 2008.

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

[Lutz, 2008] C. Lutz. The complexity of conjunctive query answering in expressive description logics. In *Proc. IJCAR 2008*, LNAI 5195, pp. 179–193. Springer, 2008.

[Ortiz *et al.*, 2008a] 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.

[Ortiz *et al.*, 2008b] M. Ortiz, M. Šimkus, and T. Eiter. Conjunctive query answering in $\mathcal{SH}$ using knots. In *Proc. DL 2008*, *CEUR Workshop Proc.*, vol 353, 2008.

[Ortiz *et al.*, 2008c] M. Ortiz, M. Šimkus, and T. Eiter. Worst-case optimal conjunctive query answering for an expressive description logic without inverses. In *Proc. AAAI'08*, pp. 504–510, 2008.

[Pratt, 1979] V.R. Pratt. Models of program logics. In *Proc. IEEE FOCS 79*, pp. 115–122, 1979.

[Sattler, 2000] U. Sattler. Description logics for the representation of aggregated objects. In *Proc. ECAI 2000*, pp. 239-243, 2000.

[Tessaris, 2001] S. Tessaris. *Questions and Answers: Reasoning and Querying in Description Logic*. PhD thesis, Univ. Manchester, CS Dept, April 2001.