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

COMPLEXITY OF CONJUNCTIVE QUERY ANSWERING IN DESCRIPTION LOGICS WITH TRANSITIVE ROLES

Thomas Eiter Carsten Lutz Magdalena Ortiz

Mantas Šimkus

INFSYS RESEARCH REPORT 1843-08-09 (PRELIMINARY)

SEPTEMBER 2008

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

W I E N

INFSYS RESEARCH REPORT
INFSYS RESEARCH REPORT 1843-08-09 (PRELIMINARY), SEPTEMBER 2008

COMPLEXITY OF CONJUNCTIVE QUERY ANSWERING IN DESCRIPTION LOGICS WITH TRANSITIVE ROLES

Thomas Eiter,[1] Carsten Lutz,[2] Magdalena Ortiz,[3] Mantas Šimkus[4]

Abstract. Answering conjunctive queries over knowledge bases in Description Logics (DLs) has received increasing attention in the last years. In the present paper, we study the computational complexity of deciding conjunctive query entailment in expressive DLs that support transitive roles and role hierarchies, but no inverse roles. We show that the problem is 2-EXPTIME-hard for the DL $\mathcal{SH}$; combining this with the known matching upper bound, we thus precisely characterize the complexity of the problem for $\mathcal{SH}$. This result extends to richer classes of DLs and queries. Our result complements the previous result proving that inverse roles make conjunctive query answering hard, showing that role hierachies in combination with transitive roles have the same effect.

[1]Institute of Information Systems, Vienna University of Technology, Austria. E-mail: eiter@kr.tuwien.ac.at.
[2]Fachbereich Informatik,Universität Bremen, Germany. E-mail: clu@informatik.uni-bremen.de
[3]Institute of Information Systems, Vienna University of Technology. E-mail: ortiz@kr.tuwien.ac.at.
[4]Institute of Information Systems, Vienna University of Technology. E-mail: simkus@kr.tuwien.ac.at.

Contents

1 Introduction

The recent use of Description Logics (DLs) in a widening range of applications has led to the study of new reasoning problems. In particular, answering queries over semantically enhanced data schemas expressed by means of DL ontologies plays an important role in areas like data and information integration, peer-to-peer data management, and ontology-based data access.

In the last years, many authors have proposed algorithms for answering (extensions of) *conjunctive queries (CQs)* over knowledge bases in various DLs and aimed at characterizing the computational complexity of this problem. A large share of this research has focused on *very expressive* DLs which contain at least the full DL $\mathcal{ALC}$ (with arbitrary TBoxes), for which the satisfiability problem is EXPTIME-hard.

The most expressive such DLs for which conjunctive query answering was shown to be decidable are $\mathcal{ALCQIb}_{reg}$ [1], $\mathcal{SHIQ}$ [3], $\mathcal{SHOQ}$ [4] and $\mathcal{ALCHOI}$ [7]. Respective algorithms yielded 2-EXPTIME upper bounds (w.r.t. the size of the knowledge base and the query) in the best case, leaving significant gaps w.r.t. the (best) EXPTIME lower bounds that are inherited from the satisfiability problem.[1] It was then shown in [6] that the problem is 2-EXPTIME-hard for $\mathcal{ALCI}$, i.e., the extension of $\mathcal{ALC}$ with inverse roles; hence, the algorithms in [1] and [3] are in fact worst case optimal. Most recently, [9] and [6] provided algorithms for the case without inverse and transitive roles that work in single exponential time. They are worst case optimal and establish EXPTIME-completeness of conjunctive query answering for $\mathcal{ALCH}$ and $\mathcal{ALCHQ}$.

However, the precise complexity of the problem remained open for expressive DLs that support transitive roles but no inverses, such as $\mathcal{SH}$, $\mathcal{SHOQ}$ and $\mathcal{ALCQb}_{reg}$. In this paper, we show that CQ answering in any DL that contains $\mathcal{SH}$, and hence in the three aforementioned DLs, is 2-EXPTIME-hard.[2] This matches the upper bounds known from [1, 3, 4, 8] and shows that transitive roles and role hierarchies make deciding conjunctive query entailment harder than satisfiability testing.

2 Preliminaries

In this section, we briefly recall knowledge bases in the DL $\mathcal{SH}$ and the problem of answering conjunctive queries over them. For the proofs, we also recall (alternating) Turing Machines and introduce notation.

2.1 Conjunctive Query Answering in $\mathcal{SH}$

$\mathcal{SH}$ Knowledge Bases. We assume countably infinite sets $\mathbf{C}$, $\mathbf{R}$ and $\mathbf{I}$ of *concept names*, *roles*, and *individuals*, respectively, where $\mathbf{C}$ contains $\top$ and $\bot$. *Concepts* are inductively defined as follows: (a) each $A \in \mathbf{C}$ 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, a, b be individuals, and let A be a concept name. Then expressions

- $C \sqsubseteq D$ are *general concept inclusions (GCIs)*;
- $r \sqsubseteq s$ are *role inclusions*;
- *Trans*(r) are *transitivity axioms*;

[1] 2-EXPTIME membership stems from [1, 3, 4], while [7] yields only a 3NEXPTIME upper bound that is believed to be not tight.
[2] $\mathcal{ALCQb}_{reg}$ can simulate transitive roles and role hierarchies (using regular expressions and role conjunction); it is strictly more expressive than $\mathcal{SHQ}$.

- $a{:}A$ and $\langle a, b\rangle{:}r$ are *assertions*.

An $\mathcal{SH}$ *knowledge base* (KB) is a tuple $\mathcal{K}{=}\langle\mathcal{T}, \mathcal{A}\rangle$, where

- $\mathcal{T}$ (the *TBox*) is a finite set of GCIs, RIs and transitivity axioms, and
- $\mathcal{A}$ (the *ABox*) is a finite set of assertions.

By $\sqsubseteq^*_{\mathcal{T}}$ we denote the reflexive transitive closure of $s \sqsubseteq r \in \mathcal{T}$.

We assume that the reader is familiar with the standard semantics of $\mathcal{SH}$ (see, e.g., [7, 11]). In what follows, we use $\mathcal{I}$ to denote an interpretation for a KB, $\Delta^{\mathcal{I}}$ for its domain, and $C^{\mathcal{I}}$ and $r^{\mathcal{I}}$ for the interpretation of a concept C and of a role r, respectively.

Conjunctive Query Answering. We assume a countably infinite set $\mathbf{V}$ of *variables*. A *conjunctive query* (CQ) over a KB $\mathcal{K}$ is a finite set of atoms of the form $A(x)$ or $r(x, y)$, where $x, y \in \mathbf{V}$, while A is a concept name and r is a role, both occurring in $\mathcal{K}$.[3]

For a CQ q over $\mathcal{K}$, let $\mathbf{V}(q)$ denote the variables occuring in q. A *match for q in an interpretation $\mathcal{I}$ for $\mathcal{K}$* is a mapping $\pi : \mathbf{V}(q) \to \Delta^{\mathcal{I}}$ such that (i) $\pi(x) \in A^{\mathcal{I}}$ for each $A(x) \in q$, and (ii) $\langle\pi(x), \pi(y)\rangle \in r^{\mathcal{I}}$ for each $r(x, y) \in q$. 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 $\mathcal{K} \models q$ holds.

Tree Model Property. The following property of $\mathcal{SH}$ KBs will be useful. An interpretation $\mathcal{I}$ is *tree-shaped*, if there is a bijection f from $\Delta^{\mathcal{I}}$ into the set of nodes of a tree $\mathbf{T}$ such that $(d, d') \in s^{\mathcal{I}}$, for any role name s, implies that there are $d_1, \ldots, d_n$ in $\Delta^{\mathcal{I}}$ and a sequence of nodes $t_1, \ldots, t_n$ in $\mathbf{T}$ such that $d = d_1$, $d' = d_n$ and for each i, $1 \leq i < n$, t_i is the father of t_{i+1}, $f(d_i) = t_i$ and $(d_i, d_{i+1}) \in r^{\mathcal{I}}$ for some transitive $r \sqsubseteq^*_{\mathcal{T}} s$. The proof of the following result is standard, using unraveling of non-tree-shaped models.

Lemma 2.1 *Suppose that $\mathcal{K}$ is an $\mathcal{SH}$ KB in which only one individual occurs. Then for every conjunctive query q, $\mathcal{K} \not\models q$ implies that $\mathcal{K}$ has some tree-shaped model $\mathcal{I}$ such that $\mathcal{I} \not\models q$.*

As $\mathcal{K} \models q$ clearly implies that $\mathcal{I} \models q$ for all tree-shaped models $\mathcal{I}$ of $\mathcal{K}$, this lemma allows us to consider only tree-shaped interpretations when deciding conjunctive query entailment.

2.2 Alternating Turing Machines

The main result of this paper relies on a reduction of the word problem for alternating Turing machines (ATMs) with exponential work space, whose definition we briefly recall; see e.g., [2] for background and details.

An ATM is given by a tuple $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, \delta)$, where

- $Q = Q_\exists \uplus Q_\forall \uplus \{q_a\} \uplus \{q_r\}$, the set of *states*, consists of *existential states* in $Q_\exists$, *universal states* in $Q_\forall$, an *accepting state* q_a, and a *rejecting state* q_r;
- Σ is the *input alphabet*;
- Γ is the *work alphabet* that contains the *blank symbol* $\llcorner\!\lrcorner$ and satisfies $\Sigma \subseteq \Gamma$;

[3] Note that no individuals occur in q; they can be simulated in the usual way. We consider only Boolean CQs (i.e., with no answer variables), to which CQs with answer variables can be reduced in the usual way.

- $q_0 \in Q_\exists \cup Q_\forall$ is the *starting* state; and
- $\delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{L, R\}$ is the *transition relation*.

Without loss of generality, we assume that $\delta(q_r, a) = \emptyset$ for all $a \in \Gamma$. For later use, we define $\delta(q, \sigma) := \{(q', \sigma', M) \mid (q, \sigma, q', \sigma', M) \in \delta\}$.

A *configuration* of $\mathcal{M}$ is a word wqw' with $w, w' \in \Gamma^*$ and $q \in Q$, whose intended meaning is that the one-side infinite tape contains the string ww' with only blanks behind it, that the machine is in state q, and that the head is on the symbol just after w. The *successor configurations* of a configuration wqw' are defined in terms of δ as usual; without loss of generality, we assume that $\mathcal{M}$ is well-behaved and never attempts to move left if the head is on the left-most position. A *halting configuration* is of the form wqw' where $q \in \{q_a, q_r\}$.

A *computation* of an ATM $\mathcal{M}$ on a word w is a sequence of configurations $K_0, K_1, \ldots$ such that $K_0 = q_0 w$ (the *initial configuration*) and K_{i+1} is a successor configuration of K_i, for all $i \geq 0$. For our concerns, we may assume that all computations are finite (on any input), and define acceptance only for this case.

A configuration $wq_a w'$ is *accepting*, if either (a) $q = q_a$, or (b) $q \in Q_\exists$ and at least one of its successor configurations is accepting, or (c) $q \in Q_\forall$ and all of its successor configurations are accepting. The ATM $\mathcal{M}$ *accepts* the input w, if the *initial configuration* is accepting. The *word problem of* $\mathcal{M}$ is, given $\mathcal{M}$ and w, to decide whether $\mathcal{M}$ accepts w. We use the following lemma.

Lemma 2.2 ([2]) *There is an ATM $\mathcal{M}$ for which the word problem is* 2-EXPTIME*-hard such that $\mathcal{M}$ works in exponential space, i.e., all configurations $w'qw''$ in computations on w fulfill $|w'w''| \leq 2^{p(|w|)}$ for some polynomial $p(n)$, and each computation of $\mathcal{M}$ on w has length at most $2^{2^{q(|w|)}}$, for some polynomial $q(n)$.*

3 2-EXPTIME-completeness of CQs in $\mathcal{SH}$

In this section, we establish the main result of this paper, viz. that CQ entailment in $\mathcal{SH}$ is 2-EXPTIME-complete.

Theorem 3.1 *The CQ entailment problem $\mathcal{K} \models q$ is* 2-EXPTIME*-complete for the DL $\mathcal{SH}$.*

The membership part follows from a number of papers (e.g., [1, 3, 4, 8]), and it thus remains to show the hardness part. We do this by a reduction from the word problem of an ATM as in Lemma 2.2, where we build on [6] by adapting a similar reduction given there.

Given $\mathcal{M}$ and a word w, we describe a KB $\mathcal{K}_w = \langle \mathcal{A}_w, \mathcal{T}_w \rangle$ and a query q_w that are constructible in polynomial time such that $\mathcal{K}_w \models q_w$ iff $\mathcal{M}$ does not accept w; since 2-EXPTIME is closed under complement, this proves 2-EXPTIME-hardness. In what follows, let $m = p(|w|)$.

Recall that each run of an ordinary (non-alternating) Turing machine is a sequence of its configurations. In case of Alternating Turing machines, this can be generalized to trees, where nodes are configurations, and branching is caused by universal states. The idea is to build $\mathcal{K}_w$ in such a way that its (relevant) models, called *computation tree models* (or *computation trees*), capture the tree-shaped structure of computations of $\mathcal{M}$ on w. From each model $\mathcal{I}$ of $\mathcal{K}_w$ such that $\mathcal{I} \not\models q_w$, it is possible to extract a computation tree model and, in turn, an accepting computation of $\mathcal{M}$ on w. On the other hand, each accepting computation corresponds to a model of $\mathcal{K}_w$ that is a counter-model for q_w. Since the size of the configurations to be represented can be exponential in m, $\mathcal{K}_w$ encodes each of them by means of the exponentially many nodes of a tree whose depth is linear in m. Hence, every computation tree is composed of *configuration trees* T of depth m, each of which represents a configuration K' of $\mathcal{M}$ that is stored via its leaves, and its root is connected to the roots of the trees representing the successor configurations of K (see Figure 1). In fact, each T stores two configurations. It uses a set of E_p *nodes* to store a *previous* configuration K, and a set of E_h *nodes* to store a *current* configuration K' that results from K by a transition of $\mathcal{M}$. The query q_w serves to check whether corresponding configurations in successive configuration trees T and T' (i.e., the current configuration in T and the previous one in T') coincide. More precisely, q_w will have a match in the computation tree $\mathcal{I}$ if this correspondence fails for some T and T' (meaning that the previous configuration is either different or not a valid configuration); such an $\mathcal{I}$ is *improper* (i.e., contains an "error"). Overall, $\mathcal{K}_w$ will entail q_w iff there is no proper computation tree that represents an accepting computation, i.e., $\mathcal{M}$ does not accept w.

In the rest of this section, we first describe the knowledge base $\mathcal{K}_w$, present then the query q_w, and finally argue about the correctness of the reduction.

3.1 Knowledge base $\mathcal{K}_w$

We define

$$\mathcal{K}_w = \langle \{a : I\}, \mathcal{T}_w \rangle$$

where a is an individual, I is a concept name (that identifies the initial node), and the TBox $\mathcal{T}_w$ contains the axioms described below.

Building configuration trees. The first set of axioms constructs configuration trees T, i.e., binary trees of depth m whose leaves are labeled with m-bit addresses (identifying the tape cells) that are implemented using the concept names $A_1, \ldots, A_m$. They are built using a role s and a concept name R for identifying their roots. For simplicity, the $m{+}1$ levels of a tree T are identified with concept names $L_0, \ldots, L_m$. For two concepts C and D, we use $C \to D$ as a shorthand for the concept $\neg C \sqcup D$. We introduce the following axioms, which generate an address bit by bit:

$$\begin{array}{rcll} R & \sqsubseteq & L_0 & \\ L_i & \sqsubseteq & \exists s.(L_{i+1} \sqcap A_{i+1}) \sqcap \exists s.(L_{i+1} \sqcap \neg A_{i+1}) & \text{for all } 0 \leq i < m \\ L_i \sqcap A_j & \sqsubseteq & \forall s.(L_{i+1} \to A_j) & \text{for all } 0 < j \leq i < m \\ L_i \sqcap \neg A_j & \sqsubseteq & \forall s.(L_{i+1} \to \neg A_j) & \text{for all } 0 < j \leq i < m \end{array}$$

Each L_m node has two successors labeled E, called E *nodes*; one is also labeled E_p (for *previous*) and called E_p *node*, and the other one E_h (for *here*) and called E_h *node*. They will be used to represent two configurations in T: the E_h-nodes for the current one, referred to as $K_h(T)$, and E_p for a possible predecessor configuration from which the current one results by a transition of $\mathcal{M}$, referred to as $K_p(T)$. The existence of these nodes is ensured by the following axiom:

$$L_m \quad \sqsubseteq \quad \exists s.(E_p \sqcap E) \sqcap \exists s.(E_h \sqcap E)$$

In the leftmost configuration tree of Figure 1, the E-nodes below one L_m node are shown.

Representing configurations inside configuration trees. As already mentioned, the configuration $K_s(T)$ of a configuration tree T, $s \in \{p, h\}$, is represented using labels of the E_s nodes in T. Each E_s-node n corresponds to one cell c_j of the tape of $\mathcal{M}$, whose address j is the address stored with $A_1, \ldots, A_m$ at its L_m parent. We store at the node n the contents of c_j and whether the head of $\mathcal{M}$ is at position j or not. To this end, we use the symbols from Γ, the states from Q and nil as concept names.[4] We label every E-node

[4] The concept nil is not needed, but simplifies matters.

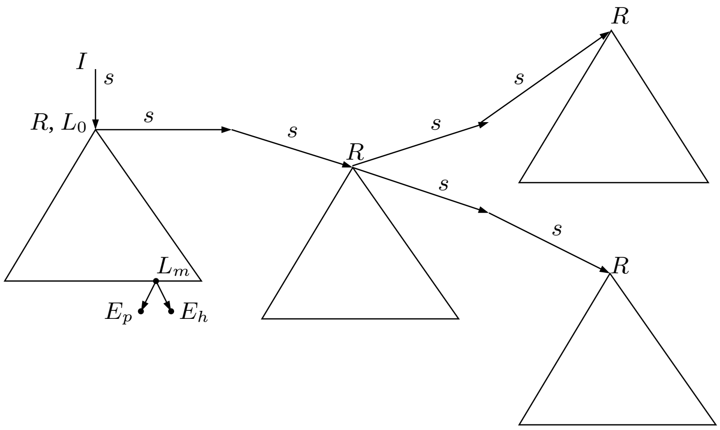

Figure 1: Some configuration trees in a computation tree

with exactly one concept from Γ (the contents of c_j), and with exactly one concept from $Q^+ := Q \cup \{nil\}$; intuitively, the label $q \in Q$ means that the head of $\mathcal{M}$ is at the tape position j and that $\mathcal{M}$ is in state q, while the label nil means that the head is not at position j:

$$\begin{aligned} E &\sqsubseteq \bigsqcup_{a\in\Gamma} a \sqcap \bigsqcap_{a\neq a'\in\Gamma} \neg(a \sqcap a') \\ E &\sqsubseteq \bigsqcup_{q\in Q^+} q \sqcap \bigsqcap_{q\neq q'\in Q^+} \neg(q \sqcap q'). \end{aligned}$$

We also call the unique pair (a, q) such that $a \sqcap q$ is true the one *stored* at an E-node. As for the configuration $K_h(T)$ represented by the E_h nodes of T, we ensure that a state $q \in Q$ is stored at exactly one bit address h, representing the correct head position. To achieve this, we use a concept name H (for the head position) and make sure that it occurs in the label of an L_m node iff its address its h, and that only an E_h successor of such an L_m node contains labels from Q.

$$\begin{aligned} L_0 &\sqsubseteq H \\ (L_i \sqcap H) &\sqsubseteq (\forall s.((L_{i+1} \sqcap A_i) \to H) \sqcap \forall s.((L_{i+1} \sqcap \neg A_i) \to \neg H)) \\ &\quad \sqcup (\forall s.((L_{i+1} \sqcap \neg A_i) \to H) \sqcap \forall s.((L_{i+1} \sqcap A_i) \to \neg H)) \quad \text{for all } 0 \leq i < m \\ (L_i \sqcap \neg H) &\sqsubseteq (\forall s.(L_{i+1} \to \neg H) \quad \text{for all } 1 \leq i < m \\ L_m \sqcap H &\sqsubseteq \forall s.(E_h \to \bigsqcup_{q\in Q} q) \\ L_m \sqcap \neg H &\sqsubseteq \forall s.(E_h \to nil) \end{aligned}$$

For the configuration $K_p(T)$ represented by the E_p nodes of T, we omit here adding similar axioms. Indeed, the query q_w that we construct will, as a byproduct, also check whether there is exactly one address such that the corresponding E_p node of T is labeled with a state $q \in Q$.[5] This is actually relevant only when T is not the initial configuration tree, and is done for any such T by comparing its E_p nodes with E_h nodes of its predecessor tree.

[5]We note that, although the E_p and E_h nodes below a given address need not be unique, the query ensures that if there are multiple nodes with the same address, they store the same (a, q) values.

Generating the computation tree. We have shown how configurations are represented inside configuration trees. Now we define axioms which ensure that configuration trees conform to computation trees that describe full computations of $\mathcal{M}$.

In the following, we use $\forall s^i.C$ to denote the i-fold nesting $\forall s.\cdots\forall s.C$. In particular, $\forall s^0.C$ is C.

Initial configuration tree. To ensure that the initial configuration tree describes the initial configuration of $\mathcal{M}$, let $w = a_0,\ldots,a_n$, let q_0 be the initial state and set:

$$\begin{array}{lcll}
I & \sqsubseteq & \exists s.R & \\
I & \sqsubseteq & \forall s^{m+1}.(\mathsf{pos}=i \rightarrow \forall s.(E_h \rightarrow a_i)) & \text{for all } i<n \\
I & \sqsubseteq & \forall s^{m+1}.(\mathsf{pos}=0 \rightarrow \forall s.(E_h \rightarrow q_0)) & \\
I & \sqsubseteq & \forall s^{m+1}.(\mathsf{pos}\geq n \rightarrow \forall s.(E_h \rightarrow \sqcup)) &
\end{array}$$

where $(\mathsf{pos}=i)$ and $(\mathsf{pos}\geq n)$ are the obvious (Boolean) concepts expressing that the value of the address $A_1,\ldots,A_m$ equals i and is at least n, respectively (recall that $\sqcup$ is the blank symbol).

Successor configuration trees. If a configuration tree T represents a configuration $K = w_0qw_1$ where $q \in Q_\exists$ is existential, then T will be linked in a proper computation tree to some configuration tree T' representing a successor configuration of K; if $q \in Q_\forall$ is universal, then T will be linked to such a configuration tree for each successor configuration of K.

To this end, we add axioms to $\mathcal{K}_w$ which state that K has, depending on whether q is existential or universal, the necessary successor configurations according to the transition relation. That the successor trees are indeed proper (and thus the computation tree is proper) will be checked using the query q_w.

In detail, to represent that T' is a successor of T upon taking the transition $(q',a',M) \in \delta(q,a)$, we label the root of T' with the concept name $T_{q',a',M}$ and we connect T to T' via two consecutive s arcs. Furthermore, if q is existential, we enforce that some T' exists with suitable label $T_{q',a',M}$ at the root, and if q is universal, we enforce that for each $(q',a',M) \in \delta(q,a)$ some T' exists with label $T_{q',a',M}$ at the root; we exploit that the state q and the symbol a are stored in an E_h of T, for one unique address.

$$\begin{array}{lcll}
R \sqcap \exists s^{m+1}.(E_h \sqcap q \sqcap a) & \sqsubseteq & \bigsqcup_{(q',a',M)\in\delta(q,a)} \exists s^2.(R \sqcap T_{q',a',M}) & \text{for all } q \in Q_\exists, a \in \Gamma, \\
R \sqcap \exists s^{m+1}.(E_h \sqcap q \sqcap a) & \sqsubseteq & \bigsqcap_{(q',a',M)\in\delta(q,a)} \exists s^2.(R \sqcap T_{q',a',M}) & \text{for all } q \in Q_\forall, a \in \Gamma.
\end{array}$$

In Figure 1, the leftmost configuration tree represents the initial configuration. Assuming that the initial state q_0 is existential, it needs to have just one successor configuration tree. The latter has two successor configuration trees, which corresponds to branching at a universal state.

Ensuring accepting computations. Since all computations of $\mathcal{M}$ are terminating and $\delta(q_r,a) = \emptyset$ for all $a \in \Gamma$, we can easily enforce that all computations are accepting by ensuring that the state q_r is never encountered:

$$q_r \sqsubseteq \bot$$

Transitions within configuration trees. We next ensure that the configuration $K_h(T)$ results from the configuration $K_p(T)$ by taking the transition that is described by the label $T_{q',a',M}$ at the root of T.

To facilitate this, we introduce two additional concept names S_ℓ and S_r that distinguish left and right successors in a configuration tree. Note that an L_i node, $1 \leq i \leq m$, is a right successor, if it is labeled with

A_i, and is a left successor otherwise. Thus, we add for all $1 \leq i \leq m$ the axioms:

$$\begin{aligned} L_i \sqcap A_i &\sqsubseteq S_r \\ L_i \sqcap \neg A_i &\sqsubseteq S_\ell \end{aligned}$$

In the following, we use $\exists(r;C)^n.D$ to denote the n-fold composition

$$\exists r.(C \sqcap \exists r.(C \sqcap \cdots (C \sqcap \exists r.D)) \cdots),$$

and similarly $\forall(r;C)^n.D$ to denote the n-fold composition

$$\forall r.(C \to \forall r.(C \to \cdots (C \to \forall r.D)) \cdots).$$

Note that $\exists(r;C)^0.D = \forall(r;C)^0.D = D$.

We locally implement the transition described by a marker $T_{q,',a',M}$, in two steps:

1. Let j be the position of the head in $K_p(T)$ (given by the address of an E_p-node that is labeled with some state $q \in Q$). The head writes a' at position j; thus in $K_h(T)$ cell c_j has contents a', and the E_h node with address j is labeled with a'. The head then moves from j one cell in the direction given by M to the cell $j\pm1$ in $K_h(T)$. Thus, the E_h node with address $j\pm1$ is labeled with q'.

   We first label the L_m node at position j with an auxiliary concept name H_p:

   $$L_m \sqcap \exists s.\big(E_p \sqcap (\bigsqcup_{q \in Q} q)\big) \quad \sqsubseteq \quad H_p$$

   Next we add, for all $q' \in Q$, $a' \in \Gamma$, $M \in \{L, R\}$, and $0 \leq i < m$ the axioms:

   $$T_{q',a',M} \sqsubseteq \forall s^m.(L_m \to T_{q',a',M})$$

   $$L_m \sqcap T_{q',a',R} \sqcap H_p \sqsubseteq \forall s.(E_h \to a')$$

   $$\begin{aligned} &L_i \sqcap \exists s.(S_\ell \sqcap \exists(s;S_r)^{m-(i+1)}.(L_m \sqcap T_{q',a',R} \sqcap H_p)) \\ &\qquad \sqsubseteq \forall r.(S_r \to \forall(r;S_\ell)^{m-(i+1)}.\forall s.(E_h \to q')) \end{aligned}$$

   $$\begin{aligned} &L_i \sqcap \exists s.(S_r \sqcap \exists(s;S_\ell)^{m-(i+1)}.(L_m \sqcap T_{q',a',L} \sqcap H_p)) \\ &\qquad \sqsubseteq \forall r.(S_\ell \to \forall(r;S_r)^{m-(i+1)}.\forall s.(E_h \to q')) \end{aligned}$$

   We exploit here that $\mathcal{M}$ never moves off the tape. To grasp the second and the third axiom, note that any two L_m-nodes n and n' in a configuration tree with stored addresses j and $j+1$, respectively, have some L_i-node n'' as common ancestor such that (*i*) n is reachable from n'' by first traveling to the left and then $m-(i+1)$ times to the right; and (*ii*) n' is reachable from n'' by first traveling to the right and then $m-(i+1)$ times to the left.

2. All remaining tape cells do not change, i.e., contain in $K_h(T)$ the same symbol as in $K_p(T)$:

   $$L_m \sqcap \exists s.(E_p \sqcap a \sqcap nil) \sqsubseteq \forall s.(E_h \to a) \quad \text{for all } a \in \Gamma.$$

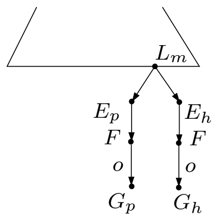

Figure 2: Extended configuration trees

The tree-shaped models of the knowledge base that we constructed so far almost correspond to accepting computations of $\mathcal{M}$ on w. In particular, if there exists an accepting run of $\mathcal{M}$ on w, then $\mathcal{K}_w$ has a model that precisely reflects this run (and can be easily constructed from it). However, the converse is not true in general, since the properness and alignment of configurations in successive configuration trees is not guaranteed: while the axioms guarantee that $K_h(T')$ is a legal successor configuration of the configuration $K_p(T')$, there is no guarantee that the E_p nodes of T' represent in fact correctly a configuration $K_p(T')$ and that, if this case, $K_p(T')$ coincides with the configuration $K_h(T)$ in the predecessor tree T of T'.

Let us call a computation tree *proper*, if for all configuration trees T and their successors T' in it $K_h(T)$ and $K_p(T')$ coincide. This property will be eventually checked using the query q_w, which will have a match if some $K_h(T)$ and $K_p(T')$ do not coincide (in particular, this will be the case if $K_p(T')$ is not a valid configuration). To this end, we extend configuration trees with auxiliary node levels.

Extending configuration trees. To enable the comparison of configurations with the query q_w, we extend configuration trees as follows. To every E_s node, for $s \in \{h, p\}$, we add a successor labeled F (called F *node*), for which a further successor labeled G_s (called G_s *node* or G *node*) is generated using a new role name o (see Figure 2):

$$\begin{array}{rcl} E_p & \sqsubseteq & \exists s.(F \sqcap \exists o.G_p) \\ E_h & \sqsubseteq & \exists s.(F \sqcap \exists o.G_h) \end{array}$$

Intuitively, the query q_w will compare all E_h nodes in the tree T with the corresponding E_p nodes in the successor tree T'. The F-nodes serve to construct a gadget that allows the query to compare the addresses and labels of E nodes on a bitwise (i.e., concept by concept) basis, while the G-nodes serve to ensure that all considered bits are from the same E_h node and E_p node, respectively.

To identify E_h nodes in T and corresponding E_p nodes in T', each E node gets a copy of the address $A_1, \ldots, A_m$ stored at its parent L_m, while its F successor gets a copy of the inverted address (obtained by bitwise complementation). We use m fresh concepts $B_1, \ldots, B_m$ for these copies and add for each $1 \leq i \leq m$ the following axioms:

$$\begin{array}{rcl} L_m \sqcap A_i & \sqsubseteq & \forall s.B_i \\ L_m \sqcap \neg A_i & \sqsubseteq & \forall s.\neg B_i \\ E \sqcap B_i & \sqsubseteq & \forall s.\neg B_i \\ E \sqcap \neg B_i & \sqsubseteq & \forall s.B_i \end{array}$$

This ensures that two E nodes n and n' store the same address iff for each concept B_i, $1 \leq i \leq m$, it holds that either (a) both n and n' are labeled with B_i, or (b) the F-successors of both n and n' are labeled with B_i. Note that a B_i can never occur both in the label of an E node and of its F-successor.

To ease the comparison of the unique pairs $(a, q) \in \Gamma \times Q^+$ stored at E-nodes in our gadget, we introduce a concept name $Z_{a,q}$ for each $a \in \Gamma$ and $q \in Q^+$; the set of all such concepts is denoted $\mathbf{Z}$. Informally, $Z_{a,q}$

represents that (a, q) is *not* the pair stored there (we use negation for technical reasons that become clear later). Now at the E_h nodes, we introduce the $Z_{a,q}$ labels as described:

$$E_h \sqsubseteq (a \sqcap q) \leftrightarrow \neg Z_{a,q} \quad \text{for all } a \in \Gamma, q \in Q^+$$

For our gadget, rather than at the E_p nodes themselves, we install the $Z_{a,q}$ labels *at their F-successors*:

$$E_p \sqsubseteq (a \sqcap q) \rightarrow \forall s.(\neg Z_{a,q} \sqcap \bigsqcap_{(a,q)\neq(a',q')} Z_{a',q'}) \quad \text{for all } a \in \Gamma \text{ and } q \in Q^+$$

Finally, we label all E_p nodes and all F-successors of all E_h nodes with all concepts in **Z**.

$$\begin{array}{lcll} E_h & \sqsubseteq & \forall s.Z_{a,q} & \text{for all } a \in \Gamma, q \in Q^+ \\ E_p & \sqsubseteq & Z_{a,q} & \text{for all } a \in \Gamma, q \in Q^+ \end{array}$$

This labeling has the following property. Let n_h be an E_h node and n_p an E_p node (supposed to be in a successor tree), and let $n_h.s$ (resp. $n_p.s$) be the F-successor of n_h (resp. n_s). If n_h stores (a, q), then n_h will be labeled with $\mathbf{Z} \setminus \{Z_{a,q}\}$, and if n_p stores (a', q'), then $n_p.s$ will be labeled with $\mathbf{Z} \setminus \{Z_{a',q'}\}$. Now suppose we compare n_h and $n_p.s$ with respect to their **Z** labels (which is equivalent to comparing the pairs (a, q) and (a', q') stored at n_h and n_p, respectively).

If they have the same labels from **Z**, then we have $Z_{a,q} = Z_{a',q'}$ and neither n_h nor $n_p.s$ is labeled with $Z_{a,q}$. Hence *both* of the following conditions *do not hold*: (*i*) n_h and n_p are labeled with $Z_{a,q}$, i.e., $Z_{a,q}$ occurs at the E-level of both gadgets; and (*ii*) $n_h.s$ and $n_p.s$ are labeled with $Z_{a,q}$, i.e., $Z_{a,q}$ occurs at the F-level of both gadgets. On the other hand, if n_h and $n_p.s$ have different labels from **Z**, then $Z_{a,q} \neq Z_{a',q'}$. In this case, for every $Z \in \mathbf{Z}$ one of the two conditions (*i*) and (*ii*) holds.

In conclusion, the **Z** labels of n_h and $n_p.s$ are different, i.e., n_h and n_p store different pairs (a, q) and (a', q'), exactly if every $Z \in \mathbf{Z}$ can be found either at the E-level of both gadgets, or at the F-level of both gadgets. This will be exploited by the query q_w.

Finally, to enable the query to match different bits at different levels, we make the role o transitive and a superrole of s:

$$s \sqsubseteq o \qquad \mathit{Trans}(o)$$

This concludes the definition of the TBox $\mathcal{T}_w$, and hence of the KB $\mathcal{K}_w$.

3.2 Query q_w

We now define the query q_w which checks whether a computation tree model $\mathcal{I}$ is proper. Recall that $\mathcal{I}$ is not proper, if some a configuration tree T in it has a successor T' such that the configuration $K_h(T)$, represented by the E_h nodes in T, is different from the configuration $K_p(T')$ represented by the E_p nodes in T' (in particular, this holds if $K_p(T')$ is not a valid configuration). The query q_w is designed to have a match in $\mathcal{I}$ precisely for such an "error" that spoils the properness of $\mathcal{I}$. More formally, we say that a computation-tree $\mathcal{I}$ has an error, if it has two nodes n_h and n_p such that:

(Q1) n_h is an E_h-node in a configuration tree T and n_p is an E_p node in a successor tree T' of T,

(Q2) n_h and n_p have the same address encoded in their labels by $B_1, \ldots, B_m$, and

(Q3) n_h and the F-successor of n_p, $n_p.s$, have different labels from **Z**.

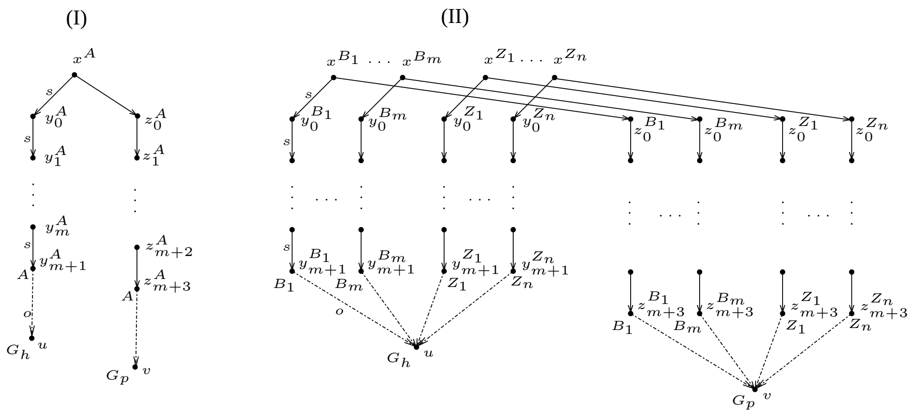

Figure 3: The basic query $q(A, u, v)$ and the final query q_w.

It is easy to see that a computation tree $\mathcal{I}$ is not proper if and only if $\mathcal{I}$ has an error; we exploit the gadget of E, F and G nodes from above to obtain a match for the query q_w if this is the case.

Informally, q_w consists of two subqueries $q_{\mathbf{B}}(u, v)$ and $q_{\mathbf{Z}}(u, v)$ which share the variables u and v that are mapped to the (unique) G-descendants of candidate nodes n_h and n_p. A match for $q_{\mathbf{B}}(u, v)$ witnesses that (Q1) and (Q2) are satisfied, while a match for $q_{\mathbf{Z}}(u, v)$ witnesses that (Q1) and (Q3) are satisfied; a combined match witnesses thus an error.

Both $q_{\mathbf{B}}(u, v)$ and $q_{\mathbf{Z}}(u, v)$ work on a bitwise (concept by concept) basis, and use the following scheme $q(A, u, v)$ that accesses two nodes in successive configuration trees that are on the same level, and tests whether they are both labeled with the concept A.

Definition 3.2 Given a concept name A and variables u, v, the query $q(A, u, v)$ is as follows:

$$\begin{aligned} q(A,u,v) \quad := \{ \; & s(x^A, y_0^A), s(y_0^A, y_1^A), \ldots, s(y_m^A, y_{m+1}^A), A(y_{m+1}^A), o(y_{m+1}^A, u), G_h(u) \\ & s(x^A, z_0^A), s(z_0^A, z_1^A), \ldots, s(z_{m+2}^A, z_{m+3}^A), A(z_{m+3}^A), o(z_{m+3}^A, v), G_p(v) \; \}. \end{aligned}$$

The query $q(A, u, v)$ is graphically shown in Figure 3(I), where solid arrows represent s-arcs and dotted arrows represent o-arcs. Informally, it works as follows. The query has two branches, a y-branch $x^A \to y_0^A \to y_1^A \cdots$ and a z-branch $x^A \to z_0^A \to z_1^A \cdots$ which have to be mapped into configuration trees T and T', respectively; as the z-branch is two arcs longer than the y branch, T' must be a successor tree of T. To map the branches into T and T', the variable x^A must be mapped either (*i*) to the root of T or (*ii*) to its parent (recall Figure 1; the root of every tree has an incoming s arc). In case (*i*), the last y_i^A variable in the y -branch, y_{m+1}^A, will be mapped to an F-node $n_h.s$ in T, and the last z_i^A variable in the z-branch, z_{m+3}^A, will be mapped to an F-node $n_p.s$ in T'; furthermore, as u and v must be mapped to G successors of $n_h.s$ respectively $n_p.s$, $n_h.s$ and $n_p.s$ must in fact be successors of an E_h-node n_h respectively an E_p-node n_p. The query checks that both $n_h.s$ and $n_p.s$ are labeled with A. In case (*ii*), the situation is similar, but y_{m+1}^A and z_{m+3}^A will be mapped one level higher up, to an E_h-node n_h in T and to an E_p-node n_p in T', respectively, provided they are both labeled with A. Since the role o is transitive and contains s, the G-node below n_h resp. n_p can be reached in one step.

Using $q(A, u, v)$ as a building block, we can now readily define the query $q_{\mathbf{B}}(u, v)$ which identifies an E_h node in a tree T and an E_p node in a successor tree T' of T that have the same address:

$$q_{\mathbf{B}}(u, v) = \bigcup_{1 \leq i \leq m} q(B_i, u, v). \tag{1}$$

Note that the sharing of the variables u and v enforces that all y-branches (resp., z-branches) end in the same node, and run through the same L_m node (recall Figure 2); this ensures that we compare all bits B_i of one address. By the labeling of the E and F-nodes, positive bits find a match at E-nodes and negative bits at the F-nodes (which carry the inverted address). A match π for $q_{\mathbf{B}}(u, v)$ in the computation tree $\mathcal{I}$ then means that at the E-predecessors of $\pi(u)$ and $\pi(v)$ the same address is encoded; only in this case such a match is possible.

The query $q_{\mathbf{Z}}(u, v)$ for checking (Q1) and (Q3) is also very simple:

$$q_{\mathbf{Z}}(u, v) = \bigcup_{Z \in \mathbf{Z}} q(Z, u, v). \tag{2}$$

To see that this query works, recall the labeling of E nodes and their F-successors with respect to **Z**. The variables y^Z_{m+1} and z^Z_{m+3} are respectively mapped either (*i*) to the F-successors of an E_h and an E_p node, or (*ii*) directly to an E_h and an E_P node in successive trees T and T'. In case (*i*), this means that both F nodes are labeled with Z, and in case (*ii*) that both E nodes are labeled with Z. If there is a match π for $q_{\mathbf{Z}}(u, v)$, then for the two gadgets containing $\pi(u)$ and $\pi(v)$, we can find each $Z \in \mathbf{Z}$ either at the E-level of both gadgets, or at the F-level of both gadgets. As discussed in the previous section, the latter holds iff the **Z** labels of the E_h-node above $\pi(u)$ and the **Z** labels of the F node above $\pi(u)$ (which is below an E_p node) are different; in other words, the E_h node and the E_p node have different labels from Σ and Q^+, and thus $K_h(T)$ and $K_P(T)$ are different.

Finally, we define q_w by joining $q_{\mathbf{B}}(u, v)$ and $q_{\mathbf{Z}}(u, v)$:

$$q_w = q_{\mathbf{B}}(u, v) \cup q_{\mathbf{Z}}(u, v). \tag{3}$$

The query q_w is graphically shown in Figure 3(II), where $\mathbf{Z} = \{Z_1, \ldots, Z_n\}$.

3.3 Entailment of q_w from $\mathcal{K}_w$

Given the construction of $\mathcal{K}_w$ and q_w above, it is not hard to argue that the problem of deciding $\mathcal{K}_w \not\models q_w$ is equivalent to verifying whether $\mathcal{M}$ accepts w, i.e., we have defined a proper reduction. Assume an arbitrary model $\mathcal{I}$ of $\mathcal{K}_w$ such that $\mathcal{I} \not\models q_w$. Since $\mathcal{K}_w$ has only one individual, by Lemma 2.1, we can w.l.o.g. assume that $\mathcal{I}$ is tree-shaped. We can further assume that $\mathcal{I}$ is a computation tree (it does not contain any labels that are not implied by the axioms). Indeed, if $\mathcal{I}$ is a tree-shaped counter model for q_w, then each sub-model of $\mathcal{I}$ (each model $\mathcal{J}$ that is homomorphically embeddable into $\mathcal{I}$) is also a counter-model for q_w. As already argued, since $\mathcal{I} \not\models q_w$, $\mathcal{I}$ is a proper computation tree, and it encodes an accepting run of $\mathcal{M}$ on w. On the other hand, given an accepting run of $\mathcal{M}$ on w, we can easily define a computation tree for which the query q_w cannot mapped because the tree does not contain errors (i.e., is proper). Hence, we conclude the following.**MS: rephrased, pls chk**

Proposition 3.3 *$\mathcal{M}$ accepts an input word w iff $\mathcal{K}_w \not\models q_w$.*

As easily verified, the knowledge base $\mathcal{K}_w$ and the query q_w are computable in polynomial time from $\mathcal{M}$ and w. This proves Theorem 3.1.

4 Related Work and Conclusion

We have shown that deciding the entailment of CQs in expressive DLs supporting transitive roles and role hierarchies is 2-EXPTIME-hard, and hence provably harder (by one exponential) than the standard reasoning tasks, like satisfiability and instance checking, in a number of DLs for which the latter problems are EXPTIME-complete.

From our proof, we obtain that CQs are 2-EXPTIME-hard even over KBs that have just a single ABox assertion, one role inclusion and one transitive role. Furthermore, since $\mathcal{SH}$ supports efficient TBox internalization [10], it extends to KBs with empty TBoxes (w.r.t. GCIs), provided that the ABox may contain complex concepts. It also extends to all expressive DLs that allow for complex role inclusions of the form $s \cdot o \sqsubseteq o'$, without the possibility to express transitivity.[6] Our proof can be easily adapted to this setting (using this inclusion and $o \sqsubseteq o'$, and by replacing in the query o by o'). Similarly, it can be adapted to role conjunction instead of a role hierarchy (by making every F node an $s \sqcap o$ successor of its E parent).

However, we point out that the interaction between, on the one hand, transitivity or role composition, and, on the other hand, role inclusion or role conjunction, is crucial in our proof and for the 2-EXPTIME-hardness result. Indeed, for expressive DLs with role hierarchies but no transitivity CQ entailment was shown to be decidable in EXPTIME [6, 9]. Further evidence of the importance of this interaction will be provided in an extended version of this report that also characterizes the complexity of $\mathcal{S}$.

In the light of our result, a natural question is under which restrictions answering CQs over $\mathcal{SH}$ knowledge bases has lower complexity. For $\mathcal{ALCI}$, it was shown that the problem becomes NEXPTIME-complete if queries are rooted, i.e., have at least one answer variable [6]. As already remarked in [6], this restriction does not reduce the worst case complexity in the presence of role hierarchies and transitivity. In fact, the query q_w in the reduction above can be easily rooted, by adding a fresh answer variable x and atoms $o(x, x^{B_1}), \ldots o(x, x^{B_m})$, $o(x, x^{Z_1}), \ldots o(x, x^{Z_n})$ that connect x to the roots of all the components of q_w.

In [9], the *order-freeness degree (OFD)* was introduced as a measure of the structural complexity of CQs, which roughly is the maximum number of query variables that reach in the query graph a common sink via a transitive role, but mutually not each other. As shown in [9], deciding entailment of CQs whose OFD is bounded by a constant from $\mathcal{SH}$ KBs is feasible in EXPTIME; unsurprisingly, q_w has unbounded OFD. As a simple consequence, all queries with constantly many variables in transitive role atoms are solvable in EXPTIME. This contrasts a very recent result of [5], which shows that CQ entailment in the DL $\mathcal{SHIQ}$ is 2-EXPTIME-hard even for queries with only two variables.

Finally, the 2-EXPTIME hardness of CQ entailment for $\mathcal{SH}$ and for $\mathcal{ALCI}$ [6] matches the known upper bounds for *unions of CQs* over $\mathcal{SHIQ}$ KBs and the even more expressive *two-way positive regular path queries* over $\mathcal{ALCQI}b_{reg}$ KBs from [1]. This shows that, once either inverse roles or role hierarchies and transitivity are allowed, one can significantly extend both the query language and the DL considered without further increasing the worst case complexity of query answering.

References

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

[6] E.g. if each role can occur only on the left hand side or only on the right hand side of inclusions.

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

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

[4] B. Glimm, I. Horrocks, and U. Sattler. Conjunctive query entailment for $\mathcal{SHOQ}$. In *Proc. of the 2007 Description Logic Workshop (DL 2007)*, volume 250 of *CEUR Electronic Workshop Proceedings*, `http://ceur-ws.org/Vol-250/`, pages 65–75, 2007.

[5] B. Glimm and Y. Kazakov. Role conjunctions in expressive description logics. Technical report, Oxford University Computing Laboratory, 2008.

[6] C. Lutz. The complexity of conjunctive query answering in expressive description logics. In A. Armando, P. Baumgartner, and G. Dowek, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR2008)*, number 5195 in LNAI, pages 179–193. Springer, 2008.

[7] M. Ortiz, D. Calvanese, and T. Eiter. Data complexity of query answering in expressive description logics via tableaux. *J. of Automated Reasoning*, 41(1):61–98, 2008. `doi:10.1007/s10817-008-9102-9`. Preliminary version available as Tech.Rep. INFSYS RR-1843-07-07, Institute of Information Systems, TU Vienna, Nov. 2007.

[8] M. Ortiz, M. Šimkus, and T. Eiter. Conjunctive query answering in $\mathcal{SH}$ using knots. In F. Baader, C. Lutz, and B. Motik, editors, *Proceedings of the 21st International Workshop on Description Logics (DL2008), May 13-16, Dresden, Germany*, volume 353 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[9] M. Ortiz, M. Šimkus, and T. Eiter. Worst-case optimal conjunctive query answering for an expressive description logic without inverses. In D. Fox and C. P. Gomes, editors, *AAAI*, pages 504–510. AAAI Press, 2008.

[10] K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI 1991)*, pages 466–471, 1991.

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