Inverse Roles Make Conjunctive Queries Hard

Carsten Lutz

Institute for Theoretical Computer Science, TU Dresden, Germany
lutz@tcs.inf.tu-dresden.de

Abstract. Conjunctive query answering is an important DL reasoning task. Although this task is by now quite well-understood, tight complexity bounds for conjunctive query answering in expressive DLs have never been obtained: all known algorithms run in deterministic double exponential time, but the existing lower bound is only an EXPTIME one. In this paper, we prove that conjunctive query answering in $\mathcal{ALCI}$ is 2-EXPTIME-hard (and thus complete), and that it becomes NEXPTIME-complete under some reasonable assumptions.

1 Introduction

When description logic (DL) knowledge bases are used in applications with a large amount of instance data, ABox querying is the most important reasoning problem. The most basic query mechanism for ABoxes is *instance retrieval*, i.e., returning all the individuals from an ABox that are known to be instances of a given query concept. Instance retrieval can be viewed as a well-behaved generalization of subsumption and satisfiability, which are the standard reasoning problems on TBoxes. In particular, algorithms for the latter can typically be adapted to instance retrieval in a straightforward way, and the computational complexity coincides in almost all cases (see [12] for an exception). In 1998, Calvanese et al. introduced *conjunctive query answering* as a more powerful query mechanism for DL ABoxes. Since then, conjunctive queries have received considerable interest in the DL community, see for example the papers [2, 3, 5–8, 11]. In a nutshell, conjunctive query answering generalizes instance retrieval by admitting also queries whose relational structure is not tree-shaped. This generalization is both natural and useful because the relational structure of ABoxes is usually not tree-shaped as well.

In contrast to the case of instance retrieval, developing algorithms for conjunctive query answering is not merely a matter of extending algorithms for satisfiability, but requires developing new techniques. In particular, all hitherto known algorithms for DLs that include $\mathcal{ALC}$ as a fragment run in deterministic double exponential runtime, in contrast to algorithms for deciding subsumption and satisfiability which require only exponential time even for DLs much more expressive than $\mathcal{ALC}$. Since the introduction of conjunctive query answering as a reasoning problem for DLs, it has remained an open question whether or not this increase in runtime can be avoided. In other words, it has not been clear whether generalizing instance retrieval to the more powerful conjunctive query answering is penalized by higher computational complexity. In this paper, we answer this question by showing that conjunctive query answering is computationally more expensive than instance retrieval when inverse roles are present. More precisely, we prove the following two results about $\mathcal{ALCI}$, the extension of $\mathcal{ALC}$ with inverse roles:

(1) Rooted conjunctive query answering in $\mathcal{ALCI}$ is co-NEXPTIME-complete, where *rooted* means that conjunctive queries are required to be connected and contain at least one answer variable. The phrase "rooted" derives from the fact that every match of such a query is rooted in at least one ABox individual. The lower bound even holds for ABoxes of the form $\{C(a)\}$ and w.r.t. empty TBoxes.

(2) Conjunctive query answering in $\mathcal{ALCI}$ is 2-EXPTIME-complete. The lower bound even holds for ABoxes of the form $\{C(a)\}$ and when queries do not contain any answer variables (or when they contain answer variables, but are not connected).

In the conference version of this paper, we will complement these results by showing that the high computational complexity of conjunctive query answering is indeed due to inverse roles. We will show that conjunctive query answering in $\mathcal{ALC}$ and $\mathcal{SHQ}$, the fragment of $\mathcal{SHIQ}$ without inverse roles, is only EXPTIME-complete. In this abstract, we concentrate on the lower bounds due to space limitations.

2 Preliminaries

We assume standard notation for the syntax and semantics of $\mathcal{ALCI}$ knowledge bases [1]. In particular, a *TBox* is a set of concept inclusions $C \sqsubseteq D$ and a *knowledge base (KB)* is a pair $(\mathcal{T}, \mathcal{A})$ consisting of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. Let $\mathsf{N_V}$ be a countably infinite set of *variables*. An *atom* is an expression $C(v)$ or $r(v, v')$, where C is an $\mathcal{ALCI}$ concept, r is a (possibly inverse) role, and $v, v' \in \mathsf{N_V}$. A *conjunctive query* q is a finite set of atoms. We use $\mathsf{Var}(q)$ to denote the set of variables occurring in the query q. Let $\mathcal{A}$ be an ABox, $\mathcal{I}$ a model of $\mathcal{A}$, q a conjunctive query, and $\pi : \mathsf{Var}(q) \to \Delta^{\mathcal{I}}$ a total function. We write $\mathcal{I} \models^{\pi} C(v)$ if $(\pi(v)) \in C^{\mathcal{I}}$ and $\mathcal{I} \models^{\pi} r(v, v')$ if $(\pi(v), \pi(v')) \in r^{\mathcal{I}}$. If $\mathcal{I} \models^{\pi} at$ for all $at \in q$, we write $\mathcal{I} \models^{\pi} q$ and call π a *match* for $\mathcal{I}$ and q. We say that $\mathcal{I}$ *satisfies* q and write $\mathcal{I} \models q$ if there is a match π for $\mathcal{I}$ and q. If $\mathcal{I} \models q$ for all models $\mathcal{I}$ of a KB $\mathcal{K}$, we write $\mathcal{K} \models q$ and say that $\mathcal{K}$ *entails* q. The *query entailment problem* is, given a knowledge base $\mathcal{K}$ and a query q, to decide whether $\mathcal{K} \models q$. This is the decision problem corresponding to query answering (which is a search problem), see e.g. [6] for details.

3 Rooted Query Entailment in $\mathcal{ALCI}$ is co-NEXPTIME-complete

Let $\mathcal{ALC}^{\mathsf{rs}}$ be the variation of $\mathcal{ALC}$ in which all roles are interpreted as reflexive and symmetric relations. Our proof of the lower bound stated as (1) above proceeds by first polynomially reducing rooted query entailment in $\mathcal{ALC}^{\mathsf{rs}}$ w.r.t. the empty TBox to rooted query entailment in $\mathcal{ALCI}$ w.r.t. the empty TBox. Then, we prove co-NEXPTIME-hardness of rooted query entailment in $\mathcal{ALC}^{\mathsf{rs}}$.

Regarding the first step, we only sketch the basic idea, which is simply to replace each symmetric role r with the composition of r^- and r. Although r is not interpreted in a symmetric relation in $\mathcal{ALCI}$, the composition of r^- and r is clearly symmetric. To achieve reflexivity, we ensure that $\exists r^-.\top$ is satisfied by all relevant individuals and for all relevant roles r. Thus, every individual can reach itself by first travelling r^- and then r, which corresponds to a reflexive loop. Since we are working without TBoxes and thus cannot use statements such as $\top \sqsubseteq \exists r^-.\top$, a careful manipulation of the ABox and query is needed. Details are given in appendix A.

Before we prove co-NExpTime-hardness of rooted query entailment in $\mathcal{ALC}^{\mathsf{rs}}$, we discuss a preliminary. An interpretation $\mathcal{I}$ of $\mathcal{ALC}^{\mathsf{rs}}$ is *tree-shaped* if there is a bijection f from $\Delta^\mathcal{I}$ into the set of nodes of a finite undirected tree (V, E) such that $(d, e) \in s^\mathcal{I}$, for some role name s, implies that $d = e$ or $\{f(d), f(e)\} \in E$. The proof of the following result is standard, using unravelling of non-tree-shaped models.

Lemma 1. *If $\mathcal{A}$ is an $\mathcal{ALC}^{\mathsf{rs}}$-ABox and q a conjunctive query, then $\mathcal{A} \not\models q$ implies that there is a tree-shaped model $\mathcal{I}$ of $\mathcal{A}$ such that $\mathcal{I} \not\models q$.*

Because $\mathcal{A} \models q$ clearly implies that $\mathcal{I} \models q$ for all tree-shaped models $\mathcal{I}$ of $\mathcal{A}$, this lemma means that we can concentrate on tree-shaped interpretations when deciding conjunctive query entailment. We will exploit this fact to give an easier explanation of the reduction that is to follow.

We now give a reduction from a NExpTime-complete variant of the tiling problem to the complement of rooted query entailment in $\mathcal{ALC}^{\mathsf{rs}}$.

Definition 1 (Domino System). *A* domino system $\mathfrak{D}$ *is a triple (T, H, V), where $T = \{0, 1, \ldots, k-1\}$, $k \geq 0$, is a finite set of* tile types *and $H, V \subseteq T \times T$ represent the* horizontal and vertical matching conditions*. Let $\mathfrak{D}$ be a domino system and $c = c_0, \ldots, c_{n-1}$ an* initial condition*, i.e. an n-tuple of tile types. A mapping $\tau : \{0, \ldots, 2^{n+1}-1\} \times \{0, \ldots, 2^{n+1}-1\} \to T$ is a* solution *for $\mathfrak{D}$ and c iff for all $x, y < 2^{n+1}$, the following holds (where $\oplus_i$ denotes addition modulo i):*

- *if $\tau(x, y) = t$ and $\tau(x \oplus_{2^{n+1}} 1, y) = t'$, then $(t, t') \in H$*
- *if $\tau(x, y) = t$ and $\tau(x, y \oplus_{2^{n+1}} 1) = t'$, then $(t, t') \in V$*
- *$\tau(i, 0) = c_i$ for $i < n$.*

For a proof of NExpTime-hardness of this version of the domino problem, see e.g. Corollary 4.15 in [9].

We show how to translate a given domino system $\mathfrak{D}$ and initial condition $c = c_0 \cdots c_{n-1}$ into an ABox $\mathcal{A}_{\mathfrak{D},c}$ and query $q_{\mathfrak{D},c}$ such that each (tree-shaped) model $\mathcal{I}$ of $\mathcal{A}_{\mathfrak{D},c}$ that satisfies $\mathcal{I} \not\models q_{\mathfrak{D},c}$ encodes a solution to $\mathfrak{D}$ and c, and conversely each solution to $\mathfrak{D}$ and c gives rise to a (tree-shaped) model of $\mathcal{A}_{\mathfrak{D},c}$ with $\mathcal{I} \not\models q_{\mathfrak{D},c}$. The ABox $\mathcal{A}_{\mathfrak{D},c}$ contains only the assertion $C_{\mathfrak{D},c}(a)$, with $C_{\mathfrak{D},c}$ a conjunction $C^1_{\mathfrak{D},c} \sqcap \cdots \sqcap C^7_{\mathfrak{D},c}$ whose conjuncts we define in the following. For convenience, let $m = 2n+2$. The purpose of the first conjunct $C^1_{\mathfrak{D},1}$ is to enforce a binary tree of depth m whose leaves are labelled with the numbers $0, \ldots, 2^m - 1$ of a binary counter implemented by the concept names $A_0, \ldots, A_{m-1}$. We use concept names $L_0, \ldots, L_m$ to distinguish the different levels of the tree. This is necessary because we work with reflexive and symmetric roles. In the following $\forall s^i.C$ denotes the i-fold nesting $\forall s. \cdots \forall s.C$. In particular, $\forall s^0.C$ is C.

$$
\begin{aligned}
C^1_{\mathfrak{D},c} := L_0 \sqcap \bigsqcap_{i<m} \forall s^i.\big(L_i \to \big(\exists s.(L_{i+1} \sqcap A_i) \sqcap \exists s.(L_{i+1} \sqcap \neg A_i)\big)\big) \sqcap \\
\bigsqcap_{i<m} \forall s^i.\bigsqcap_{j<i} \big((L_i \sqcap A_j) \to \forall s.(L_{i+1} \to A_j) \sqcap \\
(L_i \sqcap \neg A_j) \to \forall s.(L_{i+1} \to \neg A_j)\big)
\end{aligned}
$$

From now on, leafs in this tree are called L_m-nodes. Intuitively, each L_m-node corresponds to a position in the $2^{n+1} \times 2^{n+1}$-grid that we have to tile: the counter A_x realized by the concept names $A_0, \ldots, A_n$ binarily encodes the horizontal position, and the counter A_y realized by $A_{n+1}, \ldots, A_m$ encodes the vertical position. We now extend the tree with some additional nodes. Every L_m-node gets three successor nodes labelled with F, and each of these F-nodes has a successor node labelled G. To distinguish the three different G-nodes below each L_m-node, we additionally label them with the concept names G_1, G_2, G_3.

$$C^2_{\mathfrak{D},c} := \forall s^m.\big(L_m \to \big(\bigsqcap_{1 \leq i \leq 3} \exists s.(F \sqcap \exists s.(G \sqcap G_i))\big)\big)$$

We want that each G_1-node represents the grid position identified by its ancestor L_m-node, the sibling G_2 node represents the horizontal neighbor position in the grid, and the sibling G_3-node represents the vertical neighbor.

$$\begin{aligned} C^3_{\mathfrak{D},c} := \forall s^m.\big(L_m \to \big(& \bigsqcap_{i \leq n} \big((A_i \to \forall s^2.(G_1 \sqcup G_3 \to A_i)) \sqcap \\ & \qquad (\neg A_i \to \forall s^2.(G_1 \sqcup G_3 \to \neg A_i))\big) \sqcap \\ & \bigsqcap_{n < i < m} \big((A_i \to \forall s^2.(G_1 \sqcup G_2 \to A_i)) \sqcap \\ & \qquad (\neg A_i \to \forall s^2.(G_1 \sqcup G_2 \to \neg A_i))\big) \sqcap \\ & E_2 \sqcap E_3 \big)\big) \end{aligned}$$

where E_2 is an $\mathcal{ALC}$-concept ensuring that the A_x value at each G_2-node is obtained from the A_x-value of its G-node ancestor by incrementing modulo 2^{n+1}; similarly, E_3 expresses that the A_y value at each G_3-node is obtained from the A_y-value of its G-node ancestor by incrementing modulo 2^{n+1}. It is not hard to work out the details of these concepts, see e.g. [10] for more details. The *grid representation* that we have enforced is shown in Figure 1. To represent tiles, we introduce a concept name D_i for each $i \in T$ and put

$$C^4_{\mathfrak{D},c} := \forall s^{m+2}.\big(G \to \big(\bigsqcup_{i \in T} D_i \sqcap \bigsqcap_{i,j \in T, i \neq j} \neg(D_i \sqcap D_j)\big)\big)$$

The initial condition is easily guaranteed by

$$C^5_{\mathfrak{D},c} := \bigsqcap_{i < n} \forall s^{m+2}.\big(\,\big(\bigsqcap_{j \leq n, \mathsf{bit}_j(i)=0} \neg A_j \sqcap \bigsqcap_{j \leq n, \mathsf{bit}_j(i)=1} A_j \sqcap \bigsqcap_{n < j < m} \neg A_j\big) \to T_{c_i}\big),$$

where $\mathsf{bit}_j(i)$ denotes the value of the j-th bit in the binary representation of i. To enforce the matching conditions, we proceed in two steps. First we ensure that they are satisfied locally, i.e., among the three G-nodes below each L_m-node:

$$\begin{aligned} C^6_{\mathfrak{D},c} := \forall s^{m+2}.\big(L_m \to \big(& \bigsqcap_{i \in T} \big(\exists s^2.(G_1 \sqcap D_i) \to \forall s^2.(G_2 \to \bigsqcup_{(i,j) \in H} D_j)\big) \sqcap \\ & \bigsqcap_{i \in T} \big(\exists s^2.(G_1 \sqcap D_i) \to \forall s^2.(G_3 \to \bigsqcup_{(i,j) \in V} D_j)\big)\big)\big) \end{aligned}$$

Second, we enforce the following condition, which together with local satisfaction of the matching conditions ensures their global satisfaction:

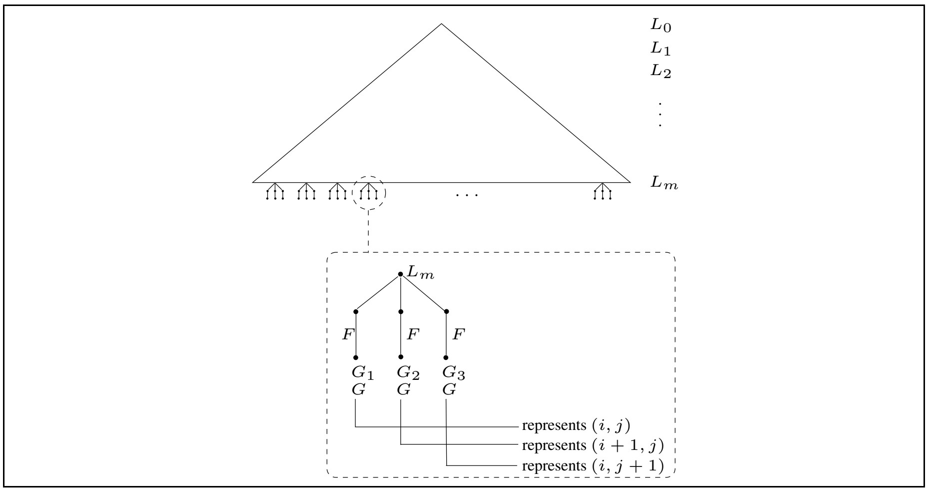

Fig. 1. The structure encoding the $2^{n+1} \times 2^{n+1}$-grid.

$(*)$ if the A_x and A_y-values of two G-nodes coincide, then their tile types coincide.

In $(*)$, a G-node can by any of a G_1-, G_2-, or G_3-node. To enforce $(*)$, we use the query. Before we give details, let us finish the definition of the concept $C_{\mathfrak{D},c}$. The last conjunct $C^7_{\mathfrak{D},c}$ enforces two technical conditions that will be explained later: if d is an F-node and e its G-node successor, then

(T1) d and e are labelled dually regarding A_i, $\neg A_i$ for all $i < m$, i.e., d satisfies A_i iff e satisfies $\neg A_i$;

(T2) d and e are labelled dually regarding $D_0, \dots, D_{k-1}$, i.e., for all $j < k$, if d satisfies D_j, then e satisfies $D_0, \dots, D_{j-1}, \neg D_j, D_{j+1}, \dots, D_{k-1}$.

We use the following concept:

$$\begin{aligned} C^7_{\mathfrak{D},c} := \forall s^{m+1}.\big(F \rightarrow \big(\bigsqcap_{i<m} & (A_i \rightarrow \forall s.(G \rightarrow \neg A_i)) \sqcap \\ & (\neg A_i \rightarrow \forall s.(G \rightarrow A_i)) \sqcap \\ \bigsqcap_{i \in T} & \exists s.(G \sqcap D_i) \rightarrow (\neg D_i \sqcap \bigsqcap_{j<k, j \neq i} D_i) \big)\big) \end{aligned}$$

We now construct the query $q_{\mathfrak{D},c}$ that does *not* match the grid representation iff $(*)$ is satisfied. In other words, $q_{\mathfrak{D},c}$ matches the grid representation if there are two G-nodes that agree on the value of the counters A_x and A_y, but are labelled with different tile types. Because of Lemma 1, we can concentrate on the grid representation as shown in Figure 1 while constructing $q_{\mathfrak{D},c}$, and need not worry about models in which domain elements that are different in Figure 1 are identified.

The construction of $q_{\mathfrak{D},c}$ is in several steps, starting with the query $q^i_{\mathfrak{D},c}$ on the left-hand side of Figure 2, where $i \in \{0, \dots, m-1\}$. In the queries $q^i_{\mathfrak{D},c}$, all the edges

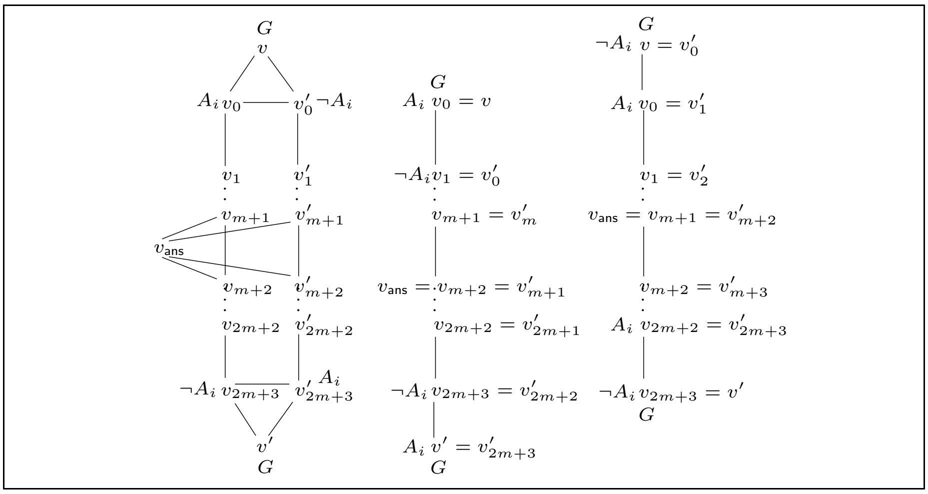

Fig. 2. The query $q^i_{\mathfrak{D},a}$ (left) and two of its collapsings (middle and right).

represent the role s and v_{ans} is the only answer variable. The edges are undirected because we are working with symmetric roles. Formally,

$$\begin{aligned} q^i_{\mathfrak{D},c} := \{\ & s(v_{i,0}, v_{i,1}), \ldots, s(v_{i,2m+2}, v_{i,2m+3}), \\ & s(v'_{i,0}, v'_{i,1}), \ldots, s(v'_{i,2m+2}, v'_{i,2m+3}), \\ & s(v_{i,0}, v'_{i,0}), s(v_{i,2m+3}, v'_{i,2m+3}), \\ & s(v, v_{i,0}), s(v, v'_{i,0}), \\ & s(v', v_{i,2m+3}), s(v', v'_{i,2m+3}), \\ & s(v_{\mathsf{ans}}, v_{i,m+1}), s(v_{\mathsf{ans}}, v_{i,m+2}), s(v_{\mathsf{ans}}, v'_{i,m+1}), s(v_{\mathsf{ans}}, v'_{i,m+2}), \\ & G(v), G(v'), A_i(v_{i,0}), \neg A_i(v'_{i,0}), \neg A_i(v_{i,2m+3}), A_i(v'_{i,2m+3})\ \} \end{aligned}$$

Observe that we dropped the index "i" to variables in Figure 2. Also observe that all the queries $q^i_{\mathfrak{D},c}$, $i < m$, share the variables v, v', and v_{ans}.

The purpose of the query $q^i_{\mathfrak{D},a}$ is to relate any two G-nodes that agree on the value of the concept name A_i. To explain how this works, we need a few preliminaries. First, a *cycle* in a query is a sequence of distinct nodes $v_0, \ldots, v_{n-1}$ such that $n \geq 2$, and $s(v_i, v_{i+1}) \in q$ or $s(v_{i+1}, v_i) \in q$ for all $i < n$, where $v_n := v_0$. A query q' is a *collapsing* of a query q if q' is obtained from q by identifying variables. Each match of $q^i_{\mathfrak{D},c}$ in our *tree-structured* grid representation gives rise to a collapsing of $q^i_{\mathfrak{D},c}$ that does not comprise any cycles. To explain how $q^i_{\mathfrak{D},c}$ works, it is helpful to analyze its cycle-free collapsings. We start with the two cycles v, v_0, v'_0 and v', v_{2m+3}, v'_{2m+3}. For eliminating each of these, we have two options:

- to remove the upper cycle, we can identify v with v_0 or v'_0;
- to remove the lower cycle, we can identify v' with v_{2m+3} or v'_{2m+3}.

Observe that if we identify v_0 and v'_0 (or v_{2m+3} and v'_{2m+3}) to collapse the cycle, there will be no matches of the query in any model.

Together, this gives four options for removing the two mentioned length-three cycles. However, two of these options are ruled out because the resulting collapsings have no match in the grid representation. The first such case is when we identify v with v_0 and v' with v_{2m+3}. Then v_0 and v_{2m+3} have to satisfy G. To continue our argument, we make a case distinction on the two options that we have for eliminating the cycle $\{v_{\mathsf{ans}}, v_{m+1}, v_{m+2}\}$.

Case (1). If we identify v_{ans} and v_{m+1}, the path from the G-variable v_0 to v_{ans} is only of length $m+1$. In our grid representation, all paths from a G-node to an ABox individual (i.e., the root) are of length $m+2$, so there can be no match of this collapsing.

Case (2). If we identify v_{ans} and v_{m+2}, the path from v_{ans} to the G-variable v_{2m+3} is only of length $m+1$ and again there is no match.

We can argue analogously for the case where we identify v with v'_0 and and v' with v'_{2m+3}. Therefore, the two remaining collapsings for eliminating the cycles $\{v, v_0, v'_0\}$ and $\{v', v_{2m+3}, v'_{2m+3}\}$ are the following:

(a) identify v with v_0 and v' with v'_{2m+3};
(b) identify v with v'_0 and v' with v_{2m+3}.

In the first case, we further have to identify v_{ans} with v_{m+2} and v'_{m+1}, for otherwise we can argue as above that there is no match. In the second case, we have to identify v_{ans} with v_{m+1} and v'_{m+2}. After this has been done, there is only one way to eliminate the cycle $v = v_0, \ldots, v_{2m+3}, v' = v'_{2m+3}, \ldots, v'_0$ such that the result is a chain of length $2m+4$ with the G-variables at both ends and the answer variable exactly in the middle (any other way to collapse means that there are no matches). The reflexive loops at the endpoints of the resulting chain and at v_{ans} can simply be dropped since we work with reflexive roles. The resulting cycle-free queries are shown in the middle and right part of Figure 2.

Note that the middle query has A_i at both ends of the chain, and the right one has $\neg A_i$ at the ends. According to our above argumentation, the original query $q^i_{\mathfrak{D},c}$ has a match in the grid representation iff one of these two collapsings has a match. Thus, every match π of $q^i_{\mathfrak{D},c}$ in the grid representation is such that $\pi(v)$ and $\pi(v')$ are (not necessarily distinct) instances of G that agree on the value of A_i. Informally, we say that $q^i_{\mathfrak{D},c}$ connects G-nodes that have the same A_i-value.

At this point, a technical remark is in order. Observe that the two relevant collapsings of $q^i_{\mathfrak{D},c}$ are such that the nodes next to the outer nodes are labelled dually w.r.t. A_i compared to the outer nodes. This is an artifact of query construction and cannot be avoided. It is the reason for introducing the F-nodes into our grid representation, and for ensuring that they satisfy Property (T1) from above.

Now set $q_{\mathsf{cnt}} := \bigcup_{i<m} q^i_{\mathfrak{D},c}$. It is easy to see that q_{cnt} connects G-nodes that have the same A_i-value, for *all* $i < m$. The query q_{cnt} is almost the desired query $q_{\mathfrak{D},c}$. Recall that we want to enforce Condition $(*)$ from above, and thus need to talk about tile types in the query. The query q_{tile} is given in the left-hand side of Figure 3 for the

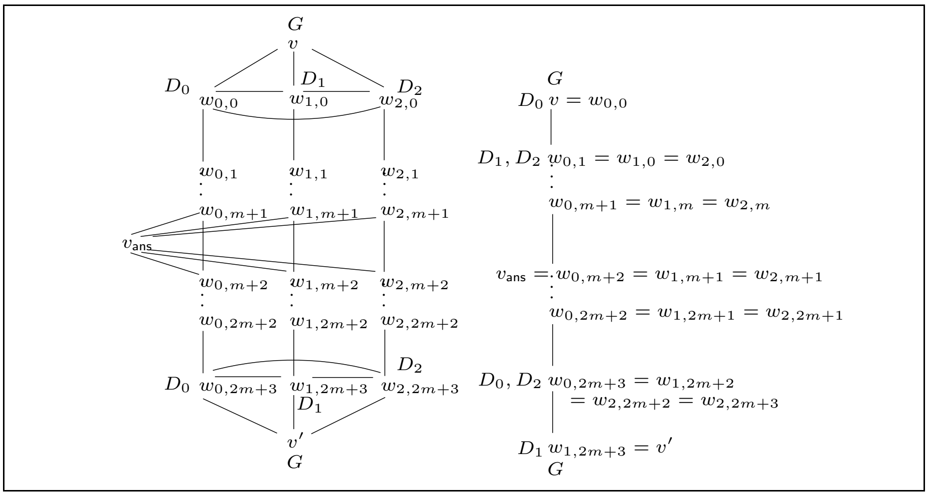

Fig. 3. The query q_{tile} (left) and one of its collapsings (right).

case of three tiles, i.e., $T = \{0, 1, 2\}$. In general, for $T = \{1, \ldots, k-1\}$, we define

$$\begin{aligned} q_{\mathsf{tile}} := \bigcup_{i<k} \{ & s(w_{i,0}, w_{i,1}), \ldots, s(w_{i,2m+2}, w_{i,2m+3}), \\ & s(w_{\mathsf{ans}}, w_{i,m+1}), s(w_{\mathsf{ans}}, w_{i,m+2}), \\ & s(v, w_{i,0}), s(v', w_{i,2m+3}), \\ & D_i(w_{i,0}), D_i(w_{i,2m+3}) \} \\ \cup \bigcup_{i<j<k} & \{ s(w_{i,0}, w_{j,0}), s(w_{i,2m+3}, w_{j,2m+3}) \} \\ \cup \ & \{ G(v), G(v') \} \end{aligned}$$

Observe that q_{cnt} and q_{tile} share the variables v, v', and v_{ans}. Also observe that q_{tile} is very similar to the queries $q^i_{\mathfrak{D},c}$, the main difference being the number of vertical chains. Whereas the queries $q^i_{\mathfrak{D},c}$ have two collapsings that are cycle-free and can have matches in the grid representation, q_{tile} has $k \cdot (k-1)$ such collapsings: for all $i, j \in T$ with $i \neq j$, there is a collapsing into a linear chain of length $2m+4$ whose end nodes are labelled D_i and D_j. An example of such a collapsing is presented on the right-hand side of Figure 3. The arguments for how to obtain these collapsing and why other collapsings have no matches in the grid representation are very similar to the line of argumentation used for $q^i_{\mathfrak{D},c}$. We only give a brief walkthrough. First, the cycle $v, w_{0,0}, \ldots, w_{k-1,0}$ can be eliminated by identifying v with one of the $w_{i,0}$. Note that we cannot eliminate the cycle by identifying all of $w_{0,0}, \ldots, w_{k-1,0}$, because then there would be no match in the grid representation. Similarly, the cycle $v', w_{0,2m+3}, \ldots, w_{k-1,2m+3}$ can be eliminated by identifying v' with one of the $w_{i,2m+3}$. We can show that $i \neq j$ by analyzing the two cases of v_{ans} being identified with $w_{i,m+1}$ or $w_{i,m+2}$. In the first case, there is no match in the grid representation because the path from v to $w_{i,m+1}$ is too short, and in the second case the same holds for the path from $w_{i,m+2}$ to v'. Thus, $i \neq j$ is shown. Also because of paths lengths, we have to identify v_{ans} with $v_{i,m+1}$ and $v_{j,m+2}$. Next, we consider the cycle $v = w_{i,0}, \ldots, w_{i,2m+3}, v' = w_{j,2m+3}, \ldots, w_{j,0}$. As in the case of q^i_w, there is only one way to eliminate this cycle such that the result is a chain of length $2m+4$ with the G-variables at both ends and the answer variable exactly in the middle, and any other way to collapse means that there are no matches. It remains to eliminate the cycles $v = w_{i,0}, \ldots, w_{i,2m+3}, v', w_{\ell,2m+3}, \ldots, w_{\ell,0}$ with $\ell \neq j$. What is important here is that we have to identify $w_{i,1}$ with $w_{\ell,0}$ and $w_{i,2m+3}$ with $w_{\ell,2m+3}$. This is the case since the alternative (identifying $w_{i,0}$ with $w_{\ell,0}$ or $v' = 2_{j,2m+2}$ with $w_{\ell,2m+3}$) leads to a variabe labelled with G, D_ℓ, and D_i (resp. D_j), and thus there is no match. Once these two identifications have been done, there is more than one way to identify the remaining nodes on the mentioned cycle, but the resulting query is always the same.

In summary, it is not hard to see that q_{tile} connects those G-nodes that are labelled by different tile types. Observe that we need property $(\mathsf{T2})$ for this query to match at all.

Now, the desired query $q_{\mathfrak{D},c}$ is simply the union of q_{cnt} and q_{tile}. From what was already said about q_{cnt} and q_{tile}, it is easily derived that $q_{\mathfrak{D},c}$ does not match the grid representation iff Property $(*)$ is satisfied. It is possible to show that there is a solution for $\mathfrak{D}$ and c iff $(\emptyset, \mathcal{A}_{\mathfrak{D},c}) \not\models q_{\mathfrak{D},c}$. We have thus proved that rooted query entailment in $\mathcal{ALCI}$ is co-NEXPTIME-hard. A matching upper bound can be obtained by adapting the techniques in [6]. More details are given in the full version of this paper.

Theorem 1. *Rooted query entailment in $\mathcal{ALCI}$ is co-*NEXPTIME*-complete. This holds even w.r.t. knowledge bases in which the TBox is empty and the ABox is a singleton.*

4 Boolean Query Entailment in $\mathcal{ALCI}$ is 2-EXPTIME-complete

If we drop the requirement that queries are connectes and have at least one answer variable, query entailment in $\mathcal{ALCI}$ becomes 2-EXPTIME-complete. An upper bound can be taken e.g. from [6]. To prove the lower bound, we again proceed in two steps: first, we polynomially reducing rooted query entailment in $\mathcal{ALC}^{\mathsf{rs}}$ w.r.t. general TBox to rooted query entailment in $\mathcal{ALCI}$ w.r.t. general TBoxes. Then, we prove 2-EXPTIME-hardness of Boolean query entailment in $\mathcal{ALC}^{\mathsf{rs}}$.

The first step is very similar to the corresponding one in Section 3. Details can be found in Appendix B. To show 2-EXPTIME-hardness of Boolean query entailment in $\mathcal{ALC}^{\mathsf{rs}}$, we reduce the word problem of exponentially space bounded alternating Turing machines (ATMs), see [4]. An *Alternating Turing Machine (ATM)* is of the form $\mathcal{M} = (Q, \Sigma, \Gamma, q_0, \Delta)$. The set of *states* $Q = Q_\exists \uplus Q_\forall \uplus \{q_a\} \uplus \{q_r\}$ 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* and Γ the *work alphabet* containing a *blank symbol* $\Box$ and satisfying $\Sigma \subseteq \Gamma$; $q_0 \in Q_\exists \cup Q_\forall$ is the *starting* state; and the *transition relation* Δ is of the form

$$\Delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{L, R\}.$$

We write $\Delta(q, \sigma)$ to denote $\{(q', \sigma', M) \mid (q, \sigma, q', \sigma', M) \in \Delta\}$ and assume w.l.o.g. that $\Delta(q_r, a) = \emptyset$ for all $a \in \Gamma$.

A *configuration* of an ATM is a word wqw' with $w, w' \in \Gamma^*$ and $q \in Q$. The intended meaning is that the one-side infinite tape contains the word ww' with only blanks behind it, the machine is in state q, and the head is on the symbol just after w. The *successor configurations* of a configuration wqw' are defined in the usual way in terms of the transition relation Δ. A *halting configuration* is of the form wqw' with $q \in \{q_a, q_r\}$.

A *computation* of an ATM $\mathcal{M}$ on a word w is a (finite or infinite) sequence of configurations $K_0, K_1, \ldots$ such that $K_0 = q_0 w$ and K_{i+1} is a successor configuration of K_i for all $i \geq 0$. The ATMs considered in the following have only *finite* computations on any input. Since this case is simpler than the general one, we define acceptance for ATMs with finite computations, only. Let $\mathcal{M}$ be such an ATM. A halting configuration is *accepting* iff it is of the form $wq_a w'$. For other configurations $K = wqw'$, acceptance depends on q: if $q \in Q_\exists$, then K is accepting iff at least one successor configuration is accepting; if $q \in Q_\forall$, then K is accepting iff all successor configurations are accepting. Finally, the ATM $\mathcal{M}$ with starting state q_0 *accepts* the input w iff the *initial configuration* $q_0 w$ is accepting. We use $L(\mathcal{M})$ to denote the language accepted by $\mathcal{M}$.

There is an exponentially space bounded ATM $\mathcal{M}$ whose word problem is 2-EXP-TIME-hard and we may assume that the length of every computation path of $\mathcal{M}$ on $w \in \Sigma^n$ is bounded by 2^{2^n}, and all the configurations wqw' in such computation paths satisfy $|ww'| \leq 2^n$, see [4]. We may also assume w.l.o.g. that $\mathcal{M}$ never attempts to move left on the left-most tape cell.

Let $w = \sigma_0 \cdots \sigma_{m-1} \in \Sigma^*$ be an input to $\mathcal{M}$. We construct an ABox $\mathcal{A}_w$, a TBox $\mathcal{T}_w$, and a query q_w such that M accepts w iff $(\mathcal{A}_w, \mathcal{T}_w) \not\models q_w$. Since a version of Lemma 1 for general TBoxes (and infinite trees) is easily established, we can concentrate on interpretations that have the shape of an (infinite) tree. Thus, tree-shaped models $\mathcal{I}$ of $\mathcal{A}_w$ and $\mathcal{T}_w$ that satisfy $\mathcal{I} \not\models q$ represent accepting computations of $\mathcal{M}$ on w and, conversely, any such computation gives rise to a tree-shaped model of $\mathcal{A}_w$ and $\mathcal{T}_w$ with $\mathcal{I} \not\models q$.

In models of $\mathcal{A}_w$ and $\mathcal{T}_w$, we represent each configuration of a computation of $\mathcal{M}$ by the leafs of a tree of depth n, very similar to the representation of the $2^{n+1} \times 2^{n+1}$-grid in Section 3. The trees representing configurations are then interconnected to a tree representing the computation of $\mathcal{M}$ on w. This situation is illustrated in Figure 4. Each of the T_i is a tree of depth n that is built using the role name s. The leafs of each such tree represent a configuration. The tree T_1 represents an existential configuration, and thus has only one successor configuration, which is represented by T_2 and connected via the same role name s also used inside the T_i trees. In contrast, the tree T_2 represents a universal configuration with two successor configurations T_3 and T_4. In the following, we will call the trees $T_1, T_2, \ldots$ *configuration trees* and the tree of interconnected configuration trees the *computation tree*.

As in Section 3, a large part of the reduction can already been done without using the query q_w. We start with this part. The ABox $\mathcal{A}_w$ is simply

$$\{a : R \sqcap I\}$$

where R is a concept name identifying the roots of configuration trees and I is a concept name identifying the root of the initial configuration. We now assemble the TBox $\mathcal{T}_w$.

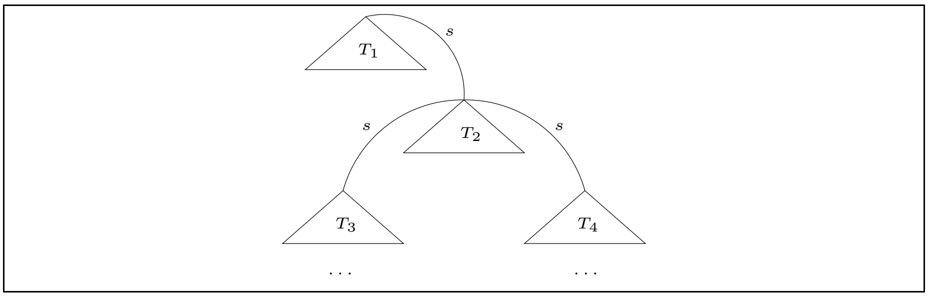

Fig. 4. Representing ATM computations.

First, we establish the configuration trees, using a construction very similar to that used in Section 3. Recall that m is the length of w and put

$$\begin{array}{rll} R \sqsubseteq L_0 & & \\ L_i \sqsubseteq \exists s.(L_{i+1} \sqcap A_i) \sqcap \exists s.(L_{i+1} \sqcap \neg A_i) & \text{for all } i < m \\ L_i \sqcap A_j \sqsubseteq \forall s.(L_{i+1} \rightarrow A_j) & \text{for } j < i < m \\ L_i \sqcap \neg A_j \sqsubseteq \forall s.(L_{i+1} \rightarrow \neg A_j) & \text{for } j < i < m \end{array}$$

As in Section 3, the leafs of the configuration trees are called L_m-nodes. Also as in that section, we add additional successors to each L_m-node: every L_m node gets two successor nodes labelled with F, and each of these F-nodes has a successor node labelled G. To distinguish the two different G-nodes below each L_m-node, we additionally label them with the concept names G_p and G_h (where h stands for *here* and p for *previous*, to be explained below). Thus, a configuration tree looks similar to what is shown in Figure 1, but with only two G-nodes below each L_m-node. The following concept inclusions generate the additional nodes and ensure that all G-nodes below an L_m-node are labelled by the concept names $A_0, \ldots, A_{m-1}$ in the same way as the L_m-node:

$$\begin{array}{rl} L_m \sqsubseteq & \big(\exists s.(F \sqcap \exists s.(G \sqcap G_h)) \sqcap \exists s.(F \sqcap \exists s.(G \sqcap G_n)) \sqcap \\ & \displaystyle\bigsqcap_{i<m} \big((A_i \rightarrow \forall s^2.(G \rightarrow A_i)) \sqcap \\ & \qquad (\neg A_i \rightarrow \forall s^2.(G \rightarrow \neg A_i))\big)\ \big) \end{array}$$

The G_h-nodes of a configuration tree describe the configuration represented by that tree, with the binary counter $A_0, \ldots, A_{m-1}$ describing the position of tape cells on the tape. The G_p-nodes describe the previous configuration in the computation (if any). To describe computations, we use the symbols from Q and Γ as concept names. Obviously, the symbols from Γ are used to represent the tape contents. The symbols from Q denote the current state and also indicate the head position. We describe some obvious facts about configurations: each tape cell is labelled with exactly one symbol and the state and head position are unique.

$$
\begin{aligned}
G &\sqsubseteq \bigsqcup_{a \in \Gamma} a \sqcap \bigsqcap_{a,a' \in \Gamma, a \neq a'} \neg(a \sqcap a') \\
G &\sqsubseteq \bigsqcap_{q,q' \in Q, q \neq q'} \neg(q \sqcap q') \\
L_0 &\sqsubseteq H \\
(L_i \sqcap H) &\sqsubseteq (\forall s.((L_{i+1} \sqcap A_i) \rightarrow H) \sqcap \forall s.((L_{i+1} \sqcap \neg A_i) \rightarrow \neg H)) \\
&\quad \sqcup (\forall s.((L_{i+1} \sqcap \neg A_i) \rightarrow H) \sqcap \forall s.((L_{i+1} \sqcap A_i) \rightarrow \neg H)) \text{ for all } i < m \\
(L_i \sqcap \neg H) &\sqsubseteq (\forall s.(L_{i+1} \rightarrow \neg H) \text{ for all } i < m \\
L_m \sqcap H &\sqsubseteq \forall s^2.(G_h \rightarrow \bigsqcup_{q \in Q} q) \\
L_m \sqcap \neg H &\sqsubseteq \forall s^2.(G_h \rightarrow \bigsqcap_{q \in Q} \neg q)
\end{aligned}
$$

Next, we describe the initial configuration. Let $w = a_0, \ldots, a_{m-1}$, q_0 be the initial state, and b the blank symbol.

$$
\begin{aligned}
I &\sqsubseteq \forall s^{m+2}.((G_h \sqcap (\mathsf{pos} = i)) \rightarrow a_i) \quad \text{for all } i < m \\
I &\sqsubseteq \forall s^{m+2}.((G_h \sqcap (\mathsf{pos} = 0) \rightarrow q_0) \\
I &\sqsubseteq \forall s^{m+2}.((G_h \sqcap (\mathsf{pos} \geq m)) \rightarrow b) \quad \text{for all } i < m
\end{aligned}
$$

Here, $(\mathsf{pos} = i)$ and $(\mathsf{pos} \geq m)$ are the obvious (Boolean) concepts expressing that the value of the counter $A_0, \ldots, A_{m-1}$ equals i and is at least m, respectively.

So far, we have been concerned with single configuration trees. Let us now turn to the *computation* tree. To enforce it, we introduce a concept name $T_{q,a,M}$ for every $q \in Q$, $a \in \Gamma$, and $M \in \{L, R\}$. If such a marker $T_{q,a,M}$ labels the root of a configuration tree T, this means that the transition q, a, M has been executed to obtain the configuration described by T. We use the following inclusions:

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

The next step is to implement the transitions described by markers locally, i.e., inside a single configuration tree with respect to the current configuration represented by the G_h-nodes and the predecessor configuration represented by the G_p-nodes. To do this, it is convenient to introduce two additional concept names S_ℓ and S_r that distinguish left successors in a configuration tree from right successors. Clearly, a node is a left successor if it is labelled L_i and $\neg A_{i+1}$ for some $i < m$, and it is a right successor if it is labelled L_i and A_{i+1}. Thus, we put for all $i < m$:

$$
\begin{aligned}
L_i \sqcap \neg A_{i+1} &\sqsubseteq S_\ell \\
L_i \sqcap A_{i+1} &\sqsubseteq S_r
\end{aligned}
$$

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

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

Note that $\exists(r;C)^0.D = \forall(r;C)^0.D = D$. Now for locally implementing the transitions described by markers. For all $q \in Q$, $a \in \Gamma$, $M \in \{L, R\}$, and $i < m$, put:

$$\begin{aligned}
&M_{q,a,M} \sqsubseteq \forall s^m.(L_m \to M_{q,a,M}) \\
&L_i \sqcap \exists s.(S_\ell \sqcap \exists(s;S_r)^{m-(i+1)}.(L_m \sqcap M_{q,a,R} \sqcap H)) \\
&\qquad\qquad \sqsubseteq \forall r.(S_r \to \forall(r;S_\ell)^{m-(i+1)}.\forall s^2.(G_h \to q)) \\
&L_i \sqcap \exists s.(S_r \sqcap \exists(s;\neg A)^{m-(i+1)}.(L_m \sqcap M_{q,a,L} \sqcap H)) \\
&\qquad\qquad \sqsubseteq \forall r.(S_\ell \to \forall(r;S_r)^{m-(i+1)}.\forall s^2.(G_h \to q))
\end{aligned}$$

We exploit here that $\mathcal{M}$ never moves left from the left-most tape cell and never right from the right-most tape cell. To understand the second and third inclusion, note that for any two L_m-nodes x and y in a configuration tree such that the value encoded by $A_0, \ldots, A_{m-1}$ at x is i and the value encoded at y is $i+1$, there exists an L_j-node z for some $j < m$ such that

- x is reachable from z by first travelling to the left successor and then $(m-j)+1$ times to the right successor;
- y is reachable from z by first travelling to the right successor and then $(m-j)+1$ times to the left successor.

We also enforce locally that tape cells which are not underneath the head do not change. Put:

$$L_m \sqcap \exists s^2.(G_p \sqcap a \sqcap \bigsqcap_{q \in Q} \neg q) \sqsubseteq \forall s^2.(G_h \to a) \quad \text{for all } a \in \Gamma$$

Since computations of $\mathcal{M}$ are terminating and $\Delta(q_r, a) = \emptyset$ for all $a \in \Gamma$, it is easy to enforce that the represented computation is accepting: simply ensure that the state q_r is never encountered:

$$q_r \sqsubseteq \bot$$

This ends the definition of the TBox $\mathcal{T}_w$. To finish the reduction, it remains to ensure the following property:

($*$) if T' is a successor configuration tree of T, x a G_h-node of T, and y a G_p-node of T' such that x and y have the same counter value regarding $A_0, \ldots, A_{m-1}$, then they are labelled identically regarding the concept names from Q and Γ.

Observe that ($*$) implies that if two G_h-nodes in the same configuration tree have the same counter value regarding $A_0, \ldots, A_{m-1}$, then they are labelled identically regarding the concept names from Q and Γ, and likewise for the G_p-nodes.

To prepare for enforcing ($*$), we introduce some additional labels that we will employ when formulating the query. First, we introduce additional concept names for a refined labelling of the roots of configuration trees:

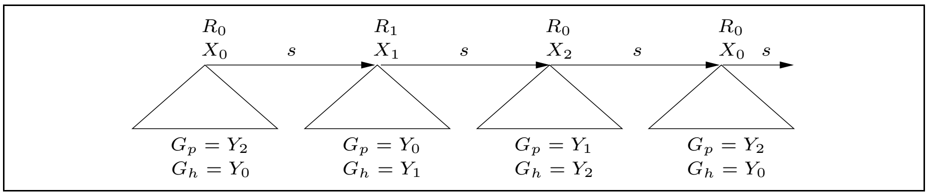

Fig. 5. The marking scheme.

- all roots of configuration trees are labelled with R_0 or R_1, alternating with each level of the computation tree;
- all roots of configuration trees are labelled with one of X_0, X_1, X_2, starting with X_0 at the root of the computation tree and then incrementing modulo 3 with each level.

This marking scheme is achieved by:

$$\begin{aligned} I &\sqsubseteq R_0 \sqcap X_0 \\ R_0 &\sqsubseteq \forall s.(R \rightarrow R_1) \\ R_1 &\sqsubseteq \forall s.(R \rightarrow R_0) \\ X_0 &\sqsubseteq \forall s.(R \rightarrow X_1) \\ X_1 &\sqsubseteq \forall s.(R \rightarrow X_2) \\ X_2 &\sqsubseteq \forall s.(R \rightarrow X_0) \end{aligned}$$

Next, we need a refined labelling of the G-nodes. According to the X_i-type of the root node, the G-nodes receive different additional labels Y_0, Y_1, Y_2:

$$\begin{aligned} X_0 &\sqsubseteq \forall s^{m+2}.((G_p \rightarrow Y_2) \sqcap (G_h \rightarrow Y_0)) \\ X_1 &\sqsubseteq \forall s^{m+2}.((G_p \rightarrow Y_0) \sqcap (G_h \rightarrow Y_1)) \\ X_2 &\sqsubseteq \forall s^{m+2}.((G_p \rightarrow Y_1) \sqcap (G_h \rightarrow Y_2)) \end{aligned}$$

The resulting scheme is illustrated in Figure 5, where we for simplicity use only existential configurations. This is justified by the fact that the labelling with R_i, X_i, and Y_i is identical for all configurations on the same level of the computation tree.

Before we can enforce $(*)$ using the query q_w, we need two additional preliminaries. First, we add a concept name $Z_{a,q}$ for each $a \in \Gamma$ and $q \in Q \cup \{\bot\}$. These concept names will be used in the query, and simply reflect the labelling with the elements of Γ and Q used as concept names:

$$G \sqsubseteq Z_{a,q} \leftrightarrow (a \sqcap q) \qquad \text{for all } a \in \Gamma \text{ and } q \in Q$$

$$G \sqsubseteq Z_{a,\bot} \leftrightarrow (a \sqcap \bigsqcap_{q \in Q} \neg q) \quad \text{for all } a \in \Gamma$$

And second, we need to ensure that if d is an F-node and e its G-node successor, then

(T1) d and e are labelled dually regarding A_i, $\neg A_i$ for all $i < m$, i.e., d satisfies A_i iff e satisfies $\neg A_i$;

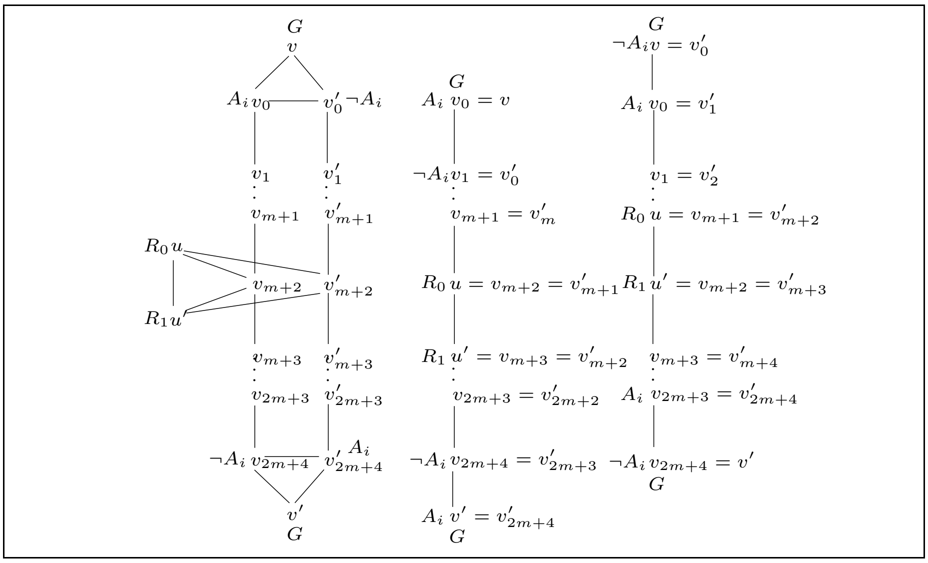

Fig. 6. The query q_w^i and two of its four relevant collapsings.

(T2) d and e are labelled dually regarding the concept names $Z_{a,q}$, $a \in \Gamma$ and $q \in Q \cup \{\bot\}$. That is, if d satisfies $Z_{a,q}$, then e satisfies $\neg Z_{a,q}$ and all $Z_{a',q'}$ with $a \neq a'$ or $q \neq q'$;

(T3) d and e are labelled dually regarding Y_0, Y_1, Y_2, i.e., for all $i < 3$, if d satisfies Y_i, then e satisfies $\neg Y_i$ and Y_j for all $j \in \{0, 1, 2\} \setminus \{i\}$.

The reason for this is exactly the same as in Section 3. We use the following implications:

$$\begin{aligned}
F \sqcap A_i &\sqsubseteq \forall s.(G \rightarrow \neg A_i) && \text{for all } i < m \\
F \sqcap \neg A_i &\sqsubseteq \forall s.(G \rightarrow A_i) && \text{for all } i < m \\
F \sqcap Z_{a,q} &\sqsubseteq \neg Z_{a,q} \sqcap \bigsqcap_{(a,q)\neq(a',q')} Z_{a',q'} && \text{for all } a \in \Gamma, q \in Q \cup \{\bot\} \\
F \sqcap Y_i &\sqsubseteq \neg Y_i \sqcap \bigsqcap_{j\in\{0,1,2\}\setminus\{i\}} Y_j && \text{for all } i < 3
\end{aligned}$$

We now construct the query q_w, which enforces $(*)$. We start with queries q_w^i, $i < m$, which are a variation of the queries $q_{\mathfrak{D},a}^i$ from Section 3. The queries q_w^i are displayed in Figure 6. As in Section 3, the purpose is to relate G-nodes that agree on the i-th bit of the counter (represented by A_i). However, there is also a notable difference: here, we want to relate G-nodes of two configuration trees T and T' such that T' is a successor configuration of T (and thus the root of T is connected to the root of T' with the role s). Compared to Figure 2, this leads to two modifications:

1. the length of the vertical chains is increased by one;

2. the answer variable v_{ans} from Figure 2 is replaced with two (non-answer) variables u and u' that are labelled with the labels of root nodes R_0 and R_1.

The reason for the first modification is to account for the additional s-edge that connects the roots of the two involved configuration trees. Since we are working with reflexive-symmetric roles, performing *only* this first modification does not suffice: it will not only allow the desired matches, but also enable matches connecting two G-nodes of the *same* configuration tree! To prevent such undesired matches, we use the variables u and u' together with the alternating labels R_0 and R_1 of roots of configuration trees. An explanation is given below. The formal definition of the query q_w^i is as follows:

$$
\begin{aligned}
q_w^i := \{\ & s(v_{i,0}, v_{i,1}), \dots, s(v_{i,2m+3}, v_{i,2m+4}), \\
& s(v'_{i,0}, v'_{i,1}), \dots, s(v'_{i,2m+3}, v'_{i,2m+4}), \\
& s(v_{i,0}, v'_{i,0}), s(v_{i,2m+4}, v'_{i,2m+4}), \\
& s(v, v_{i,0}), s(v, v'_{i,0}), \\
& s(v', v_{i,2m+4}), s(v', v'_{i,2m+4}), \\
& s(u, u'), s(u, v_{m+2}), s(u, v'_{m+2}), s(u', v_{m+2}), s(u', v'_{m+2}), \\
& G(v), G(v'), A_i(v_{i,0}), \neg A_i(v'_{i,0}), \neg A_i(v_{i,2m+4}), A_i(v'_{i,2m+4}), R_0(u), R_1(u')\ \}
\end{aligned}
$$

Observe that we dropped the index "i" to variables in Figure 6. Also observe that all the queries q_w^i, $i < m$, share the variables v, v', u, and u'.

As in Section 3, we are interested in the cycle-free collapsings of the queries q_w^i because only such collapsings can match in the (tree-shaped!) computation tree. To eliminate the cycles v, v_0, v'_0 and v', v_{2m+4}, v'_{2m+4}, it is again the case the there are only two options:

(a) identify v with v_0 and v' with v'_{2m+4};
(b) identify v with v'_0 and v' with v_{2m+4}.

Let us consider the first case. To eliminate the cycle u, u', v_{m+2}, we have four options:

(i) identify u with v_{m+1} and u' with v_{m+2};
(ii) identify u with v_{m+2} and u' with v_{m+3};
(iii) identify u' with v_{m+1} and u with v_{m+2};
(iv) identify u' with v_{m+2} and u with v_{m+3}.

Options (i) and (iii) result in R_0 or R_1 being a label of v_{m+2}. Since R_0 and R_1 are found only at the root of configuration trees, configuration trees are of depth $m+2$, and the path between $v = v_0$ and v_{m+2} is only of length $m+1$, the queries resulting from (i) and (iii) have no matches in the computation tree. Options (ii) and (iv) lead to queries that may have matches. The difference between the two resulting queries is that one is for the case where the predecessor configuration is labelled with R_0 and the successor one with R_1, and for the other query it is the other way round. In Case (b), we can argue similarly to show that only options (i) and (iii) lead to queries which may have matches.

It remains to eliminate the large cycle, which is straightforward. In summary, we can thus show that there are four cycle-free collapsings of q_w^i that may have a match in the computation tree: the two collapsings displayed on the right-hand side of Figure 6

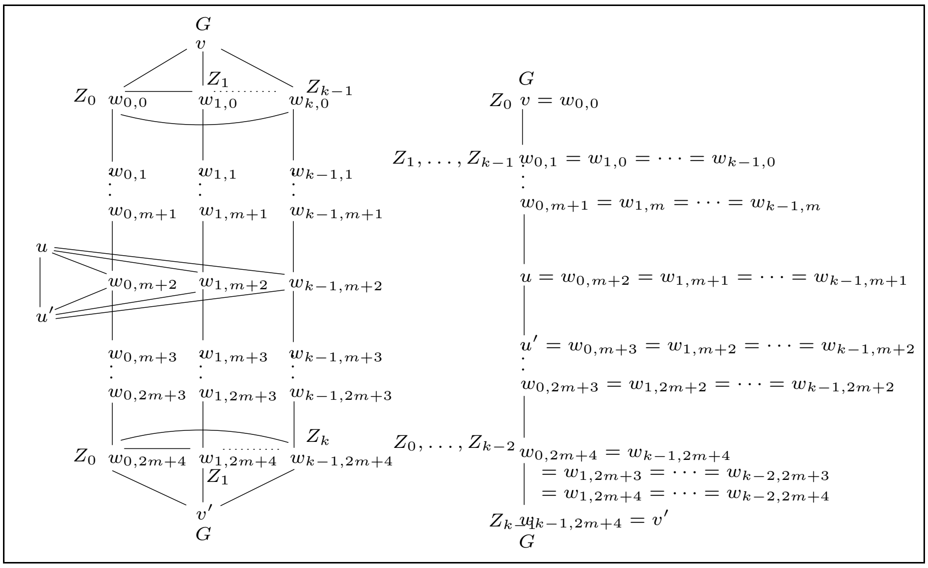

Fig. 7. The query q_Z (left) and one of its collapsings (right).

and two collapsings obtained from the displayed ones by swapping the locations of u and u' and of R_0 and R_1.

Similar to what was done in Section 3, we can argue that the query $\bigcup_{i<m} q_w^i$ connects precisely those G-nodes d and e such that d belongs to a configuration tree T, e belongs to a configuration tree T' which is a successor of T, and d and e have the same counter value regarding $A_0, \ldots, A_{m-1}$. We want to achieve $(*)$, and thus have to ensure that the query matches iff the $Z_{a,q}$-labelling of two such nodes is different. This is easily achieved by modifying the query q_{tile} from Section 3, as shown in Figure 3, according to Points 1 and 2 above. The result of this modification is displayed in Figure 7, where k denotes the cardinality of $\Gamma \times (Q \cup \{\bot\})$ and Z_i the concept name $Z_{a,q}$ such that (a, q) is the i-th element (starting with element 0) in an assumed well-order on $\Gamma \times (Q \cup \{\bot\})$. Formally, the query is defined as follows.

$$\begin{aligned} q_Z := \bigcup_{i<k} \{ & s(w_{i,0}, w_{i,1}), \ldots, s(w_{i,2m+3}, w_{i,2m+4}), \\ & s(u, w_{i,m+2}), s(u', w_{i,m+2}), \\ & s(v, w_{i,0}), s(v', w_{i,m+4}), \\ & Z_i(w_{i,0}), Z_i(w_{i,2m+4}) \} \\ \cup \bigcup_{i<j<k} & \{ s(w_{i,0}, w_{j,0}), s(w_{i,2m+4}, w_{j,2m+4}) \} \\ \cup & \{ s(u, u'), G(v), G(v') \} \end{aligned}$$

Observe that the variables v, v', u, and u' are shared with the queries q_w^i. The query q_Z has $2k \cdot (k-1)$ collapsings that may have a match in the computation tree, one of them

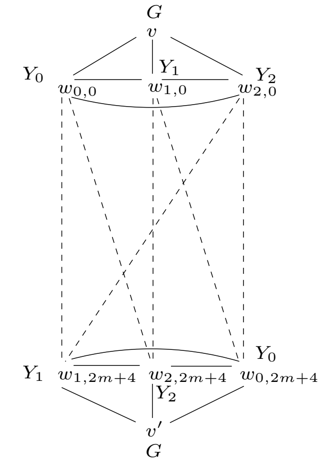

Fig. 8. The query q_Y.

shown on the right-hand side of Figure 7. The arguments are a blend of those used for q^i_w in the current section and q_{tile} in Section 3. Details are left to the reader.

Now consider the query $q = q_Z \cup \bigcup_{i<m} q^i_w$. It is almost the desired query q_w: it connects G-nodes in successor configurations that have the same counter value and a different labelling regarding the concept names $Z_{a,q}$. However, this query actually connectes *too many* G-nodes: since we work with symmetric roles, it also connects the G_h-nodes of a configuration tree with the G_p-nodes of the *predecessor* configuration tree. This is still true even if we replace the G-label of the variable v with G_h and the G-label of the variable v' with G_p.

To break this symmetry, we use the marking with concept names Y_0, Y_1, Y_2 as shown in Figure 5. Given that figure, it is not hard to see that the query q should only connect G-nodes that are labelled identically regarding Y_0, Y_1, Y_2. Thus, we need a generalization of the query q^i_w to three values (Y_0, Y_1, Y_2) instead of two (A_i and $\neg A_i$). To achieve, this, we replace the two nodes v_0 and v'_0 of q^i_w with three nodes which we label Y_0, Y_1, and Y_2, and likewise for v_{2m+4} and v'_{2m+4}. Then, we establish a vertical chain of length $2m + 4$ between the nodes with different Y_i-labels. The resulting query is sketched in Figure 8, where each dashed line indicates a sequence of length $2m+4$, as in the queries q^i_w and q_Z. The variables u and u', which are connected to the midpoint of each such sequence, are not shown to avoid cluttering the picture. Formally, we define:

$$q_Y := \bigcup_{i,j\in\{0,1,2\},i\neq j} \{s(w'_{i,j,1}, w'_{i,j,2}), \dots, s(w'_{i,j,2m+2}, w'_{i,j,2m+3}), \\ s(u, w'_{i,j,m+2}), s(u', w'_{i,j,m+2}), \\ s(w'_{i,0}, w'_{i,j,1}), s(w'_{i,j,2m+3}, w'_{j,2m+4})\}$$

$$\begin{aligned}\cup\ \{&s(v, w'_{0,0}), s(v, w'_{1,0}), s(v, w'_{2,0}),\\ &s(w'_{0,0}, w'_{1,0}), s(w'_{0,0}, w'_{2,0}), s(w'_{1,0}, w'_{2,0}),\\ &s(w'_{0,2m+4}, w'_{1,2m+4}), s(w'_{0,2m+4}, w'_{2,2m+4}), s(w'_{1,2m+4}, w'_{2,2m+4}),\\ &s(v', w'_{0,2m+4}), s(v', w'_{1,2m+4}), s(v', w'_{2,2m+4}),\\ &s(u, u'),\\ &Y_0(w'_{0,0}), Y_0(w'_{0,2m+4}), Y_1(w'_{1,0}), Y_1(w'_{1,2m+4}), Y_2(w'_{2,0}), Y_2(w'_{2,2m+4}),\\ &G(v), G(v')\}\end{aligned}$$

Observe that the variables v, v', u, and u' are shared with the other queries that we have defined so far. Argueing similar as we have done before, it is not hard to show that q_Y has six collapsings which may have matches in the computation tree. Each collapsing is a linear sequence of length $2m + 5$, with Y_i at both endpoints, for some $i \in \{1, 2, 3\}$. The two midpoints of these sequences are labelled with R_0 and R_1, in any order.

Now, the wanted query q_w is simply $q_Y \cup q_Z \cup \bigcup_{i<m} q_w^i$.

Lemma 2. *$\mathcal{M}$ accepts input w iff there is a model $\mathcal{I}$ of $\mathcal{T}_w$ and $\mathcal{A}_w$ such that $\mathcal{I} \not\models q_w$.*

We have thus proved:

Theorem 2. *Query entailment in $\mathcal{ALCI}$ is 2-*EXPTIME*-complete. This holds even for queries without answer variables and w.r.t. knowledge bases in which the ABox is a singleton.*

The proof of Theorem 2 shows that query entailment becomes 2-EXPTIME-hard if we drop the first requirement of rooted query entailment (that queries contain at least one answer variable), but not the second (that queries are connected). It is trivial to modify the proof such that it works for the case where the second requirement is dropped, but not the first. Indeed, we simply add an atom $\top(v)$ to q_w, where v is an answer variable. Observe that the resulting query is disconnected.

5 Conclusion

We have shown that in the presence of inverse roles, conjunctive query answering is computationally more costly than instance checking. A Corresponding NEXPTIME upper bound for Theorem 1 and containment of conjunctive query entailment in EXPTIME for $\mathcal{ALC}$ will be shown elsewhere. As (almost) remarked by a reviewer, the proof of Theorem 2 can easily be adapted to rooted query entailment if transitive roles and role hierarchies are present. Details on this will also be given elsewhere.

Acknowledgement We thanks the anonymous reviewers for valuable remarks on the submitted version of this paper.

References

1. F. Baader, D. L. McGuiness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook: Theory, implementation and applications*. Cambridge University Press, 2003.

2. D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proceedings of the 17th ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.
3. 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. Welty, editors, *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*. AAAI Press, 2006.
4. A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.
5. B. Glimm, I. Horrocks, and U. Sattler. Conjunctive query answering for description logics with transitive roles. In B. Parsia, U. Sattler, and D. Toman, editors, *Proceedings of the 2006 International Workshop on Description Logics (DL'06)*, volume 189 of *CEUR-WS*, 2006.
6. B. Glimm, C. Lutz, I. Horrocks, and U. Sattler. Answering conjunctive queries in the $\mathcal{SHIQ}$ description logic. In M. Veloso, editor, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 299–404. AAAI Press, 2007.
7. I. Horrocks, U. Sattler, and S. Tobies. Reasoning with individuals for the description logic SHIQ. In D. MacAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction (CADE-17)*, number 1831 in Lecture Notes in Computer Science, Germany, 2000. Springer Verlag.
8. U. Hustadt, B. Motik, and U. Sattler. Data complexity of reasoning in very expressive description logics. In L. P. Kaelbling and A. Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 466–471. Professional Book Center, 2005.
9. C. Lutz. *The Complexity of Reasoning with Concrete Domains*. PhD thesis, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 2002.
10. C. Lutz, C. Areces, I. Horrocks, and U. Sattler. Keys, nominals, and concrete domains. *Journal of Artificial Intelligence Research (JAIR)*, 23:667–726, 2005.
11. M. Ortiz, D. Calvanese, and T. Eiter. Data complexity of answering unions of conjunctive queries in $\mathcal{SHIQ}$. In B. Parsia, U. Sattler, and D. Toman, editors, *Proceedings of the 2006 International Workshop on Description Logics (DL'06)*, volume 189 of *CEUR-WS*, 2006.
12. A. Schaerf. On the complexity of the instance checking problem in concept languages with existential quantification. *Journal of Intelligent Information Systems*, 2:265–278, 1993.

A From $\mathcal{ALC}^{\mathsf{rs}}$ to $\mathcal{ALCI}$ without TBoxes

We show that rooted query entailment in $\mathcal{ALC}^{\mathsf{rs}}$ w.r.t. the empty TBox can be polynomially reduced to rooted query entailment in $\mathcal{ALCI}$ w.r.t. the empty TBox.

As already explained, the main idea behind the reduction is to replace each symmetric role r with the composition of r^- and r. Let $\mathcal{A}$ be an $\mathcal{ALC}^{\mathsf{rs}}$ ABox and q a conjunctive query. We assume w.l.o.g. that all concepts in $\mathcal{A}$ are in negation normal form (NNF), i.e., that negation is applied only to concept names. Let $\mathsf{Ind}(\mathcal{A})$ denote the set of all individual names occurring in $\mathcal{A}$, $\mathsf{rol}(\mathcal{A})$ be the set of role names used in $\mathcal{A}$, and let $\mathsf{rol}(q)$ be defined analogously. Fix a fresh concept name R. Intuitively, the purpose of R is to distinguish "real" domain elements from the auxiliary ones that serve as intermediate points in the composition of r^- and r. Also, define X as an abbreviation for $\bigsqcap_{r\in\mathsf{rol}(\mathcal{A})\cup\mathsf{rol}(q)}\exists r^-.\top$. We will enforce that X is satisfied by all relevant real individuals, thus achieving reflexivity.

We now present the details of the reduction. For each concept C in NNF, let $\delta(C)$ denote the result of replacing

- every subconcept $\exists r.C$ with $\exists r^-.\exists r.(C \sqcap R \sqcap X)$, and
- every subconcept $\forall r.C$ with $\forall r^-.\forall r.C$;

Now define an $\mathcal{ALCI}$ ABox $\mathcal{A}'$ and a query q' by manipulating $\mathcal{A}$ and q as follows:

1. replace every concept assertion $C(a) \in \mathcal{A}$ with $\delta(C)(a)$;
2. for all $a \in \mathsf{Ind}(\mathcal{A})$, add a concept assertion $R \sqcap X(a)$ to $\mathcal{A}$;
3. replace every role assertion $r(a,b) \in \mathcal{A}$ with $r(c,a)$ and $r(c,b)$, where c is a fresh individual name;
4. for every variable v in q, add $R(v)$ to q;
5. replace every role atom $r(v,v') \in q$ with $r(v^*,v)$ and $r(v^*,v)$, where v^* is a fresh variable.

The following lemma shows that our reduction is correct.

Lemma 3. *$\mathcal{A} \not\models q$ iff $\mathcal{A}' \not\models q'$.*

Proof. "$\Rightarrow$". If $\mathcal{A} \not\models q$, then there is a model $\mathcal{I}$ of $\mathcal{A}$ such that $\mathcal{I} \not\models q$. Define a model $\mathcal{I}'$ as follows:

- $\Delta^{\mathcal{I}'} = \Delta^{\mathcal{I}} \cup \{x_{d,r,e} \mid r \in \mathsf{rol}(\mathcal{A}) \cup \mathsf{rol}(q) \text{ and } (d,e) \in r^{\mathcal{I}}\}$;
- $r^{\mathcal{I}'} = \{(x_{d,r,e}, d), (x_{d,r,e}, e) \mid (d,e) \in r^{\mathcal{I}}\}$
- $A^{\mathcal{I}'} = A^{\mathcal{I}}$ for all concept names A except R;
- $R^{\mathcal{I}'} = \Delta^{\mathcal{I}}$;
- $a^{\mathcal{I}'} = a^{\mathcal{I}}$ for all $a \in \mathsf{Ind}(\mathcal{A})$;
- if c was introduced into $\mathcal{A}'$ to split the assertion $r(a,b) \in \mathcal{A}$, set $c^{\mathcal{I}'} = x_{a^{\mathcal{I}},r,b^{\mathcal{I}}}$.

It is readily checked that $\mathcal{I}'$ is a model of $\mathcal{A}'$. In particular, $X^{\mathcal{I}'} = \Delta^{\mathcal{I}'}$ since roles are interpreted reflexively in $\mathcal{I}$. Furthermore, since $\mathcal{I} \not\models q$, we have $\mathcal{I}' \not\models q'$: suppose to the contrary that $\mathcal{I}' \models^{\pi} q'$ for some match π. Since q' contains the atom $R(v)$ for every variable $v \in \mathsf{Var}(q)$, we have $\pi(v) \in \Delta^{\mathcal{I}}$ for all $v \in \mathsf{Var}(q)$. Let π' be the restriction of π to the variables in $\mathsf{Var}(q)$. It is readily checked that $\mathcal{I} \models^{\pi'} q$, which is a contradiction.

"$\Leftarrow$". If $\mathcal{A}' \not\models q'$, then there is a model $\mathcal{I}'$ of $\mathcal{A}'$ such that $\mathcal{I}' \not\models q'$. Define a model $\mathcal{I}$ as follows:

- $\Delta^{\mathcal{I}} = (R \sqcap X)^{\mathcal{I}'}$;
- $r^{\mathcal{I}} = \{(d,e) \mid \exists f.(f,d) \in r^{\mathcal{I}} \wedge (f,e) \in r^{\mathcal{I}}\}$;
- $A^{\mathcal{I}} = A^{\mathcal{I}'} \cap \Delta^{\mathcal{I}}$;
- $a^{\mathcal{I}} = a^{\mathcal{I}'}$ for all $a \in \mathsf{Ind}(\mathcal{A})$.

Observe that $r^{\mathcal{I}}$ is reflexive (due to the choice of $\Delta^{\mathcal{I}}$ as a subset of $X^{\mathcal{I}}$) and symmetric. Also observe that the interpretation of the individual names is well-defined: since $\mathcal{A}'$ contains $R \sqcap X(a)$ for all $a \in \mathsf{Ind}(\mathcal{A})$, $a^{\mathcal{I}'} \in \Delta^{\mathcal{I}}$. Since $\mathcal{I}' \not\models q'$ and it is easily seen that $\mathcal{I} \models q$ would imply $\mathcal{I}' \models q'$, we have $\mathcal{I} \not\models q$. It remains to show that $\mathcal{I}$ is a model of $\mathcal{A}$. This is a consequence of the following claim, which is easily proved by induction on the structure of C.

Claim. For all $d \in \Delta^{\mathcal{I}}$ and all $C \in \mathsf{sub}(\mathcal{A})$, $d \in \delta(C)^{\mathcal{I}'}$ implies $d \in C^{\mathcal{I}}$.

We only do the two interesting cases.

- Let $C = \forall r.D$. Then $\delta(C) = \forall r^-.\forall r.\delta(D)$. Let $(d,e) \in r^{\mathcal{I}}$. We have to show that $e \in D^{\mathcal{I}}$. Since $(d,e) \in r^{\mathcal{I}}$, by definition of $\mathcal{I}$ we have $(d,e) \in (r^-)^{\mathcal{I}'} \circ r^{\mathcal{I}'}$. Since $d \in \delta(C)^{\mathcal{I}'}$, we have $e \in D^{\mathcal{I}'}$ and it remains to apply the induction hypothesis.
- Let $C = \exists r.D$. Then $\delta(C) = \exists r^-.\exists r.(\delta(D) \sqcap R \sqcap X)$. Since $d \in \delta(C)^{\mathcal{I}'}$, there is an $e \in \Delta^{\mathcal{I}'}$ such that (i) $(d,e) \in (r^-)^{\mathcal{I}'} \circ r^{\mathcal{I}'}$ and (ii) $e \in (\delta(D) \sqcap R \sqcap X)^{\mathcal{I}'}$. By (ii), $d \in \Delta^{\mathcal{I}}$. By (i) and definition of $\mathcal{I}$, $(d,e) \in r^{\mathcal{I}}$. By (ii) and induction hypothesis, $d \in D^{\mathcal{I}}$ and we are done.

❑

B From $\mathcal{ALC}^{\mathsf{rs}}$ to $\mathcal{ALCI}$ with TBoxes

We show that rooted query entailment in $\mathcal{ALC}^{\mathsf{rs}}$ w.r.t. general TBoxes can be polynomially reduced to rooted query entailment in $\mathcal{ALCI}$ w.r.t. general TBoxes. The general strategy is as in Section A, but the presence of general TBoxes actually makes the reduction easier.

Let $\mathcal{A}$ be an $\mathcal{ALC}^{\mathsf{rs}}$ ABox, $\mathcal{T}$ a TBox, and q a conjunctive query. For each concept C, let $\delta(C)$ denote the result of replacing

- every subconcept $\exists r.C$ with $\exists r^-.\exists r.C$, and
- every subconcept $\forall r.C$ with $\forall r^-.\forall r.C$;

Now define an $\mathcal{ALCI}$ ABox $\mathcal{A}'$, TBox $\mathcal{T}'$, and a query q' by manipulating $\mathcal{A}$, $\mathcal{T}$, and q as follows, where R is a fresh concept name:

1. replace every concept assertion $C(a) \in \mathcal{A}$ with $\delta(C)(a)$;
2. replace every concept inclusion $C \sqsubseteq D \in \mathcal{T}$ with $\delta(C) \sqsubseteq \delta(D)$;

3. for all $a \in \mathsf{Ind}(\mathcal{A})$, add a concept assertion $R(a)$ to $\mathcal{A}$;
4. add the following concept inclusions to $\mathcal{T}$:

$$R \sqsubseteq \forall r^{-}.\neg R \qquad \neg R \sqsubseteq \forall r.R \qquad R \sqsubseteq \bigsqcap_{r \in \mathsf{rol}(\mathcal{A}) \cup \mathsf{rol}(\mathcal{T}) \cup \mathsf{rol}(q)} \exists r^{-}.\top$$

5. replace every role assertion $r(a, b) \in \mathcal{A}$ with $r(c, a)$ and $r(c, b)$, where c is a fresh individual name;
6. for every variable v in q, add $R(v)$ to q;
7. replace every role atom $r(v, v') \in q$ with $r(v^{*}, v)$ and $r(v^{*}, v)$, where v^{*} is a fresh variable.

The prove of the following lemma is similar to that of Lemma A. Details are left to the reader.

Lemma 4. $(\mathcal{A}, \mathcal{T}) \not\models q$ *iff* $(\mathcal{A}', \mathcal{T}') \not\models q'$.