Status $\mathcal{QIO}$: An Update

Birte Glimm[1], Yevgeny Kazakov[1], and Carsten Lutz[2]

[1] Oxford University Computing Laboratory, UK
[2] Universität Bremen, Germany

Abstract. We prove co-N2ExpTime-hardness for conjunctive query entailment in the description logic $\mathcal{ALCOIF}$, thus improving the previously known 2ExpTime lower bound. The result transfers to OWL DL and OWL2 DL, of which $\mathcal{ALCOIF}$ is an important fragment. A matching upper bound remains open.

1 Introduction

Due to its importance for ontology-based data access and data integration conjunctive query (CQ) answering has developed into one of the most widely studied reasoning tasks in description logic (DL). Nevertheless, the precise complexity (and sometimes even decidability) of CQ answering in several important expressive DLs is still an open problem. In particular, this concerns fragments of the W3C-standardized OWL DL ontology language that comprise nominals, inverse roles, and number restrictions, a combination of expressive means that is notorious for interacting in intricate ways. In this paper, we concentrate on the basic such fragment $\mathcal{ALCOIF}$ in which number restrictions take the form of global functionality constraints.

Decidability of CQ answering in $\mathcal{ALCOIF}$ and its extension $\mathcal{ALCOIQ}$ with qualified number restrictions has been shown only very recently [1]. Since the proof is based on a mutual enumeration of finite models and theorems of first-order logic, it does not yield any upper complexity bound. The best known lower bound for CQ answering in $\mathcal{ALCOIF}$ is 2ExpTime, inherited from the fragment $\mathcal{ALCI}$ of $\mathcal{ALCOIF}$ that does not include nominals and functionality constraints [2, 3]. The aim of this paper is to improve upon this lower bound by establishing co-N2ExpTime-hardness. Note that CQ answering in the fragment $\mathcal{ALCIF}$ of $\mathcal{ALCOIF}$ that does not include nominals is in 2ExpTime [4], and the same is true for the fragment $\mathcal{ALCQO}$ that does not include inverse roles [5] and $\mathcal{ALCOI}$ that does not include functionality restrictions [6]. Thus, our result shows that the combination of nominals, inverse roles, and number restrictions leads to an increase of complexity of CQ answering from 2ExpTime to (at least) co-N2ExpTime. This parallels the situation for the subsumption problem, which is co-NExpTime-complete for $\mathcal{ALCOIF}$, but ExpTime-complete in any of $\mathcal{ALCIF}$, $\mathcal{ALCQO}$, and $\mathcal{ALCOI}$. Since $\mathcal{ALCOIF}$ is a fragment of OWL DL (in both the OWL1 and the OWL2 version), our co-N2ExpTime lower bound obviously also applies to CQ answering in this language.

We prove our result by a reduction of the tiling problem that requires to tile a torus of size $2^{2^n} \times 2^{2^n}$. Our construction combines elements of two existing hardness proofs, but also requires the development of novel ideas. We follow the general strategy of the

proofs that show N2ExpTime-hardness of satisfiability in $\mathcal{SROIQ}$ [7] and in the extension of $\mathcal{SHOIF}$ with role conjunctions [8]. One central part of those proofs is the realization of a counter that counts up to 2^{2^n}. We realize this counter using a (rather subtle!) adaptation of the conjunctive queries that have been developed in [2, 3] to establish 2ExpTime-hardness of CQ-answering in $\mathcal{ALCI}$.

An extended technical report including proofs and further details is available [9].

2 Preliminaries

We assume standard notation for the syntax and semantics of $\mathcal{ALCOIF}$ knowledge bases [10]. The presence of nominals allows for only working with TBoxes, which consist of concept inclusions (CIs) $C \sqsubseteq D$. A *knowledge base (KB)* is then simply a TBox. Let N_V be a countably infinite set of variables. An atom is an expression $C(v)$ or $r(v, v')$, where C is a (potentially compound) $\mathcal{ALCOIF}$-concept, r is an atomic role, and $v, v' \in N_V$.[3] A conjunctive query q is a finite set of atoms. We use $\mathsf{Var}(q)$ to denote the set of variables that occur in the query q. Let $\mathcal{K}$ be an $\mathcal{ALCOIF}$ KB, $\mathcal{I} = (\cdot^{\mathcal{I}}, \Delta^{\mathcal{I}})$ a model of $\mathcal{K}$, q a conjunctive query, and $\pi\colon \mathsf{Var}(q) \to \Delta^{\mathcal{I}}$ 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 $\langle \pi(v), \pi(v') \rangle \in r^{\mathcal{I}}$. 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 *conjunctive 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, see e.g. [4].

A *domino system* is a triple $D = (T, H, V)$, where $T = \{1, \dots, k\}$ is a finite set of *tiles* and $H, V \subseteq T \times T$ are *horizontal* and *vertical matching relations*. A *tiling* of $m \times m$ for a domino system D with *initial condition* $c^0 = \langle t_1^0, \dots, t_n^0 \rangle$, $t_i^0 \in T$ for $1 \leq i \leq n$, is a mapping $t\colon \{0, \dots, m-1\} \times \{0, \dots, m-1\} \to T$ such that $\langle t(i, j), t(i+1 \bmod m, j) \rangle \in H$, $\langle t(i, j), t(i, j+1 \bmod m) \rangle \in V$, and $t(i, 0) = t_{i-1}^0$ $(0 \leq i, j < m)$. There exists a domino system D_0 for which it is N2ExpTime-complete to decide, given an initial condition c^0 of length n, whether D_0 admits a tiling of $2^{2^n} \times 2^{2^n}$ with initial condition c^0 [11].

3 Conjunctive Query Entailment in $\mathcal{ALCOIF}$

Our aim is to construct, for an initial condition c^0 of length n, an $\mathcal{ALCOIF}$-KB $\mathcal{K}_0$ and conjunctive query q_0 such that $\mathcal{K}_0 \not\models q_0$ iff D_0 admits a tiling of $2^{2^n} \times 2^{2^n}$ with initial condition c^0.

Intuitively, the models of $\mathcal{K}_0$ that we are interested in have the form depicted in Figure 1: a torus of dimension $2^{2^n} \times 2^{2^n}$, where the lower left corner is identified by the nominal o, the upper right corner by the nominal e, each horizontal dashed arrow denotes the role h, and each vertical dotted arrow the role v. We will install two counters that identify the vertical and horizontal position of torus nodes. To store the counter values, we use binary trees of (roughly) depth n below the torus nodes, where each

[3] Complex concepts C in atoms $C(x)$ are used w.l.o.g.; to eliminate them, we can replace $C(x)$ with $A_C(x)$ for a fresh atomic concept A_C and add $C \sqsubseteq A_C$ to the TBox.

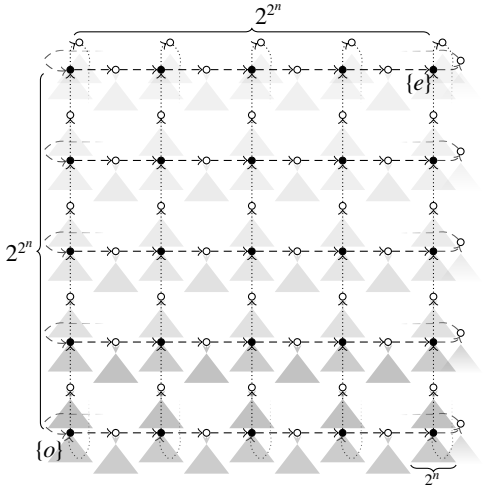

Fig. 1. Schematic depiction of the torus

of the 2^n leaves store one bit of each counter (represented via concept names X and Y). The filled circles in Figure 1 denote true torus nodes, which are labeled by a tile later on, while the unfilled circles denote auxiliary nodes that will help us in properly incrementing the counters. This incrementation is the main difficulty of the reduction, and it is achieved with the help of the query q_0. As the details are intricate, we defer a discussion of the details until later, and first concentrate on the construction of $\mathcal{K}_0$.

The following concept inclusions (1) to (9) of $\mathcal{K}_0$ lay the foundation for enforcing the torus structure with attached trees. Successors in trees are connected via the composition of the roles r^- and r, from now on denoted by $r^-; r$. This is needed in the query construction later on, similar to the use of symmetric roles in [2, 3]. We call additional nodes between r^- and r the 'intermediate' tree nodes. Note that no branching occurs at intermediate nodes. Also for the query construction, the root of a tree below a true torus node is the torus node itself while the root of a tree below an auxiliary torus node is reachable by traveling one step along the role r (see Figure 1). To distinguish these two kinds of trees, we label trees of the former kind with the concept name B and call them black trees, and trees of the latter kind with the concept name W and call them white trees. Later on, we will use white trees that are on the vertical axis to increment the vertical counter and white trees that are on the horizontal axis to increment the horizontal counter. To support this, we further label white trees of the former kind with V and white trees of the latter kind with H. The basic idea for constructing the torus itself is similar to what is done in [12, 7, 8]: the maximum value of both counters (indicated by the concept names M_X and M_Y) identifies the upper right corner, which has to satisfy the nominal e and is thus unique. Inverse functionality for h and v then guarantees uniqueness of elements for all other values of the horizontal and vertical counters, and that the torus 'closes' in the expected way. We use concept names $L_0, \dots, L_n$ to mark the levels of the trees, to deal with the symmetry of the composition $r^-; r$. Thus, the

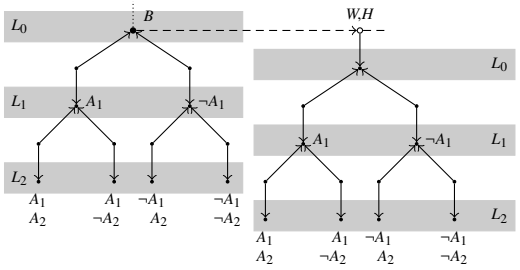

Fig. 2. A black and a white tree, for $n = 2$

concept $B \sqcap L_0$ identifies the true torus nodes.

$$\{o\} \sqsubseteq B \sqcap L_0 \tag{1}$$

$$B \sqcap L_0 \sqsubseteq \exists h.(W \sqcap \exists r.(H \sqcap W \sqcap L_0) \sqcap \exists h.(B \sqcap L_0)) \tag{2}$$

$$B \sqcap L_0 \sqsubseteq \exists v.(W \sqcap \exists r.(V \sqcap W \sqcap L_0) \sqcap \exists v.(B \sqcap L_0)) \tag{3}$$

$$B \sqcap L_0 \sqcap M_X \sqcap M_Y \sqsubseteq \{e\} \tag{4}$$

$$L_i \sqsubseteq \exists r^-.\exists r.(A_{i+1} \sqcap L_{i+1}) \sqcap \exists r^-.\exists r.(\neg A_{i+1} \sqcap L_{i+1}) \quad {\scriptstyle i<n} \tag{5}$$

$$A_i \sqcap L_j \sqsubseteq \forall r^-.\forall r.(L_{j+1} \to A_i) \qquad {\scriptstyle 1\le i\le j<n} \tag{6}$$

$$\neg A_i \sqcap L_j \sqsubseteq \forall r^-.\forall r.(L_{j+1} \to \neg A_i) \qquad {\scriptstyle 1\le i\le j<n} \tag{7}$$

$$\mathsf{C} \sqsubseteq \forall r.\mathsf{C} \sqcap \forall r^-.\mathsf{C} \qquad \text{for all } \mathsf{C} \in \{B, W, H, V\} \tag{8}$$

$$\top \sqsubseteq \leqslant 1 h^-.\top \qquad \top \sqsubseteq \leqslant 1 v^-.\top \tag{9}$$

Note that the concept names $A_1, \ldots, A_n$ implement a binary counter for the leafs of the trees, i.e., for counting the bit positions in the horizontal and vertical counters. In summary, the internal structure of the trees is as shown in Figure 2 where branching tree nodes have dark background and intermediate nodes have light background.

The next step is to make sure that the horizontal and vertical counter have value 0 at the origin and that M_X is true at the root of a tree when the horizontal counter has reached the maximum value, and similarly for M_Y. We use $\forall(r^-; r)^n.C$ to denote the $2n$-quantifier prefixed $\forall r^-.\forall r. \cdots \forall r^-.\forall r.C$. Recall that the concept name X represents the truth value of bits of the horizontal counter, and likewise for Y and the vertical counter.

$$\{o\} \sqsubseteq \forall(r^-; r)^n.(\neg X \sqcap \neg Y) \tag{10}$$

$$L_n \sqsubseteq (X \leftrightarrow M_X) \sqcap (Y \leftrightarrow M_Y) \tag{11}$$

$$L_{i-1} \sqcap \exists r^-.\exists r.(L_i \sqcap M_X \sqcap A_i) \sqcap \exists r^-.\exists r.(L_i \sqcap M_X \sqcap \neg A_i) \sqsubseteq M_X \quad {\scriptstyle 0<i\le n} \tag{12}$$

$$L_{i-1} \sqcap \exists r^-.\exists r.(L_i \sqcap M_Y \sqcap A_i) \sqcap \exists r^-.\exists r.(L_i \sqcap M_Y \sqcap \neg A_i) \sqsubseteq M_Y \quad {\scriptstyle 0<i\le n} \tag{13}$$

$$L_{i-1} \sqcap \exists r^-.\exists r.(L_i \sqcap \neg M_X) \sqsubseteq \neg M_X \quad {\scriptstyle 0<i\le n} \tag{14}$$

$$L_{i-1} \sqcap \exists r^-.\exists r.(L_i \sqcap \neg M_Y) \sqsubseteq \neg M_Y \quad {\scriptstyle 0<i\le n} \tag{15}$$

The general strategy for updating the horizontal and vertical counter is as follows. We introduce additional concept names X' and Y', which represent the truth value of the bits of two additional binary counters, the 'primed versions' of the horizontal and vertical counter. Using $\mathcal{K}_0$, we ensure that, in black trees, the X-counter has the same value

as the X'-counter, and likewise for the Y- and Y'-counter. In white trees, we distinguish between horizontal incrementation indicated by the concept name H and vertical incrementation indicated by the concept name V: if the tree satisfies H, then the value of the X'-counter is the value of the X-counter incremented by one, while the values of the Y- and Y'-counter coincide; if the tree satisfies V, it is the other way around. The remaining job to be accomplished by the query q_0 is then to

($*$) ensure that the value of the X'-counter (resp. Y'-counter) in a (black or white) tree is identical to the value of the X-counter (resp. Y-counter) in its 'successor trees', i.e., in trees that can be reached by traveling a single step in the torus along the roles h or v.

This behavior of the counters, with the exception of ($*$), is implemented by the subsequent concept inclusions. To increment a counter, we use a concept name F to mark the bits that have to be flipped. Another concept name S, which is propagated down from the root to a single leaf, is used to mark the unique bit of the incremented counter such that (i) all bits to the right are flipped from 1 to 0, (ii) the bit itself is flipped from 0 to 1, and (iii) all bits to the left remain unchanged. As a special case, all bits flip when the maximum counter value has been reached. In the following, CIs (16) to (19) implement the proper marking by F and S, CIs (21) to (23) realize the actual incrementation of the X-counter to the X'-counter in (white) H-trees, and CI (24) ensures that the Y- and Y'-counters have the same value in H-trees and in black trees. We also need CIs (21) to (24) with H replaced by V, X by Y, X' by Y', Y by X, and Y' by X'.

$$L_0 \sqcap (\neg M_X \sqcup \neg M_Y) \sqsubseteq S \tag{16}$$

$$L_0 \sqcap (M_X \sqcap M_Y) \sqsubseteq F \sqcap \neg S \tag{17}$$

$$S \sqcap L_{i-1} \sqsubseteq \forall r^-.\forall r.(L_i \to [\,((A_i \to \neg F \sqcap \neg S) \sqcap (\neg A_i \to S)) \sqcup ((A_i \to S) \sqcap (\neg A_i \to F \sqcap \neg S))\,]) \quad 0<i\leq n \tag{18}$$

$$F \sqcap \neg S \sqcap L_{i-1} \sqsubseteq \forall r^-.\forall r.(L_i \to F \sqcap \neg S) \quad 0<i\leq n \tag{19}$$

$$\neg F \sqcap \neg S \sqcap L_{i-1} \sqsubseteq \forall r^-.\forall r.(L_i \to \neg F \sqcap \neg S) \quad 0<i\leq n \tag{20}$$

$$H \sqcap L_n \sqcap F \sqcap \neg S \sqsubseteq X \sqcap \neg X' \tag{21}$$

$$H \sqcap L_n \sqcap S \sqsubseteq \neg X \sqcap X' \tag{22}$$

$$H \sqcap L_n \sqcap \neg F \sqcap \neg S \sqsubseteq (X \sqcap X') \sqcup (\neg X \sqcap \neg X') \tag{23}$$

$$(B \sqcup H) \sqcap L_n \sqsubseteq (Y \sqcap Y') \sqcup (\neg Y \sqcap \neg Y') \tag{24}$$

To enable the construction of a query q_0 that enforces ($*$), we add a further (single) $r^-;r$-successor to each leaf in each tree. At this extra node, which is marked with the concept name L_{n+1}, the truth value of all concept names A_i, X, X', Y, Y' is complemented compared to its predecessor L_n-node. We also introduce a marker concept Q that is true at the intermediate node between each L_n-node and L_{n+1}-node. This is similar to what is done in [2, 3]. We call such intermediate nodes Q-nodes.

$$L_n \sqsubseteq \exists r^-.(Q \sqcap \exists r.L_{n+1}) \tag{25}$$

$$L_n \sqcap \mathsf{C} \sqsubseteq \forall r^-.\forall r.(L_{n+1} \to \neg \mathsf{C}) \qquad L_n \sqcap \neg \mathsf{C} \sqsubseteq \forall r^-.\forall r.(L_{n+1} \to \mathsf{C}) \quad \text{for all } \mathsf{C} \in \{A_1, \ldots, A_n, X, X', Y, Y'\} \tag{26}$$

The construction of $\mathcal{K}_0$ is not yet finished. However, it will be more convenient to construct the remaining part along with the query q_0. The query is assembled from

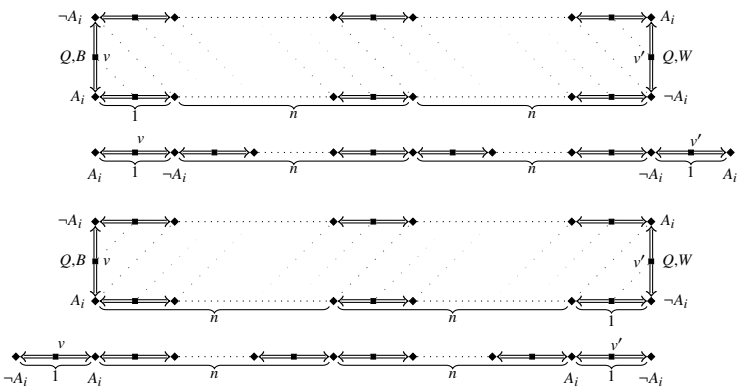

Fig. 3. A counting component of the query q_0 and the two ways to fold it

two types of components: *counting components* and *copying components*. We start with presenting and explaining a simplified version of counting components, which are then refined in a second step. The final query q_0 will contain one counting component for each bit of the counter $A_1, \ldots, A_n$ that counts the leaves of our trees. The simplified version of the counting component for A_i is shown as the topmost cycle in Figure 3, where, for the moment, every arrow should be interpreted as a role atom that uses the role name r. The goal is that matches of this query component should map (i) v to a Q-node of a black tree and v' to the Q-node of a white successor tree such that the two predecessor L_n-nodes agree on the value of A_i or, symmetrically, (ii) v' to a Q-node of a white tree and v to the Q-node of a black successor tree such that the two predecessor L_n-nodes agree on the value of A_i. By taking the union of all counting queries for $A_1, \ldots, A_n$ such that the variables v and v' are shared, we thus link leaves of successor trees that represent the same bit position for the horizontal and vertical counter, which is the first important step towards enforcing $(*)$.

Due to the Q-concept at v and v', each variable labeled with A_i or $\neg A_i$ is matched to an L_n-node or an L_{n+1}-node. Ignoring the presence of the role names h and v in the torus and pretending that white trees are rooted directly on the torus, each match of the counting component gives rise to one of the two 'foldings' presented in Figure 3. These foldings are obtained by identifying variables that are matched to the same domain element, as indicated by the dotted lines. Intuitively, the two foldings correspond to the bit A_i being false (upper folding) and true (lower folding). For brevity, we omit the concept names Q, B, W in the foldings. Since the long sides of the counting component are of length $2n + 1$ (counted in terms of compositions $r^-; r$) and trees are of depth n, the two trees involved in a match cannot be further away than one step in the torus. Due to the use of B and W, they cannot be identical.

In the discussion of the simplified counting components above, we have neglected the presence of the roles h and v in the torus that we need to 'cross' when matching the query in the described way. Refining the counting queries to deal with these roles is the major challenge in the current reduction, compared to the 2ExpTime lower bound in [2, 3] where only a single role r is used. Note that we cannot just introduce a single

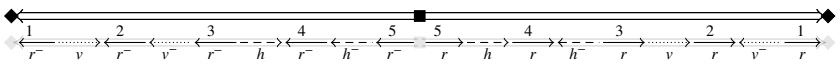

Fig. 4. Internal structure of a meta role consisting of 18 roles

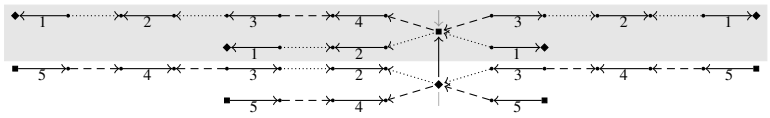

Fig. 5. Side chains for branching (upper) and intermediate (lower) nodes

h-arrow and v-arrow into the counting components since we want to match either h or v, but not both; moreover, the position of the h-arrow/v-arrow would shift back and forth with the different ways to fold the query. To solve the problem, we replace each role composition $r^-; r$ in the query (but not in the trees!) with a composition of 18 roles that we call a 'meta role', see Figure 4 where every solid edge denotes the role name r.

Note that the meta role is symmetric, like the composition $r^-; r$. The aim is that each meta-role in the refined counting query matches one $r^-; r$-role composition that connects two successor nodes in a tree. To resolve the mismatch between the role composition $r^-; r$ of length two and the meta role of length 18, the meta role is designed such that the remaining parts can be folded away into 'side chains' that we will add to the tree, i.e., chains of roles that start at each tree node. There are five ways to achieve such a folding, one for each corresponding pair of r^-- and r-arrows in the left and right half of the meta role. For example, we can use the 3rd r^--arrow and the 3rd r-arrow to match the $r^-; r$-composition in the tree, and then have to fold away the prefix composition $r^-; v; r^-; v^-$ before the 3rd r^--arrow, the infix composition $h; r^-; h^-; r^-; r; h; r; h^-$ between the 3rd r^--arrow and the 3rd r-arrow, and the postfix composition $v; r; v^-; r$ following the 3rd r-arrow. Observe that the infix composition is symmetric and thus can be folded into a chain. The postfix composition is the converse of the infix composition, which will allow us to leave a side chain that we have entered with the postfix composition using the prefix composition of the subsequent meta role. Similar foldings allow us to match the $r^-; r; h; r$-compositions required to move up one level in a black tree and then cross via an h-edge to the root of a white successor tree, the $r^-; h; r^-; r$-compositions that allows us to cross from the root of a white tree to a black tree and then move down one level, and to perform the two analogous crossing with h replaced by v.

The scheme for adding side chains is shown in Figure 5, where intermediate tree nodes (lower node on the center line) receive different chains than branching tree nodes (upper node on the center line). These chains are added to every node in the tree with the exception of the roots of black trees, as those are directly on the torus and adding side chains would violate inverse functionality of h and v. Note that the side chains attached to branching tree nodes are precisely the possible postfix compositions mentioned above, while the side chains attached to intermediate tree nodes are foldings of what we called infix compositions above. The chains are generated by the following CIs, to be added to $\mathcal{K}_0$. We use the concept $N_B = (L_0 \sqcap W) \sqcup \bigsqcup_{1 \le i \le n+1} L_i$ to identify

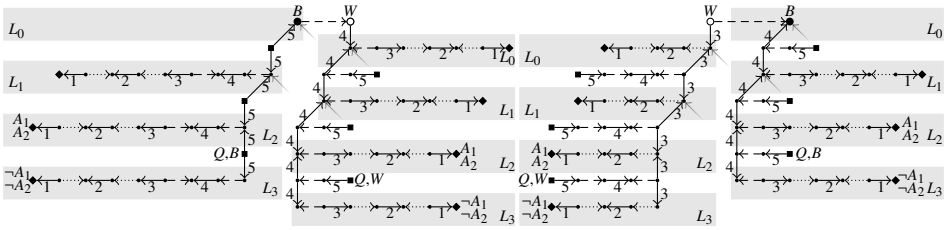

Fig. 6. From black to white trees via h (left) and from white to black trees via h (right)

branching tree nodes and $N_I = \exists r.\bigsqcup_{1\le i\le n+1} L_i$ to identify intermediate tree nodes.

$$N_B \sqsubseteq \exists h.\exists r.\exists h^-.\exists r.\exists v.\exists r.\exists v^-.\exists r.\top \sqcap \exists v.\exists r.\exists v^-.\exists r.\top \sqcap \exists h^-.\exists r.\exists v.\exists r.\exists v^-.\exists r.\top \sqcap \exists v^-.\exists r.\top \tag{27}$$

$$N_I \sqsubseteq \exists v.\exists r^-.\exists v^-.\exists r^-.\exists h.\exists r^-.\exists h^-.\exists r^-.\top \sqcap \exists h.\exists r^-.\exists h^-.\exists r^-.\top \sqcap \exists v^-.\exists r^-.\exists h.\exists r^-.\exists h^-.\exists r^-.\top \sqcap \exists h^-.\exists r^-.\top \tag{28}$$

In matches of the refined query, the endpoints of the two foldings shown in Figure 3 will match at the end of (possibly empty) side chains at the level $n+1$, v and v' will match at the end of the side chains between the levels n and $n+1$, and the adjacent inner nodes labeled with $\neg A_i$ resp. A_i will match at the end of the side chains at the level n. For this reason, we propagate all relevant concept names to the end of those chains. For $\mathsf{C} \in \{A_1, \dots, A_n, \neg A_1, \dots, \neg A_n, X, \neg X, Y, \neg Y\}$, $\mathsf{C}' \in \{Q, B, W\}$, add the concept inclusions

$$(L_n \sqcup L_{n+1}) \sqcap \mathsf{C} \sqsubseteq \forall h.\forall r.\forall h^-.\forall r.\forall v.\forall r.\forall v^-.\forall r.\mathsf{C} \sqcap \forall v.\forall r.\forall v^-.\forall r.\mathsf{C} \sqcap \forall h^-.\forall r.\forall v.\forall r.\forall v^-.\forall r.\mathsf{C} \sqcap \forall v^-.\forall r.\mathsf{C} \tag{29}$$

$$Q \sqcap \mathsf{C}' \sqsubseteq \forall v.\forall r^-.\forall v^-.\forall r^-.\forall h.\forall r^-.\forall h^-.\forall r^-.\mathsf{C}' \sqcap \forall h.\forall r^-.\forall h^-.\forall r^-.\mathsf{C}' \sqcap \forall v^-.\forall r^-.\forall h.\forall r^-.\forall h^-.\forall r^-.\mathsf{C}' \sqcap \forall h^-.\forall r^-.\mathsf{C}' \tag{30}$$

Figure 6 shows, for $n = 2$, how to fold the refined counting query such that v is mapped to a Q-node of a black tree and v' to a Q-node of a white successor tree that can be reached via crossing an h-edge in the torus, and likewise for the case where v' is mapped to a white tree, and v to a black successor tree reachable via h. We display only those side chains that are needed for accommodating the query match. To get started, note that in the left part of Figure 6, the h-edge in the right half of a meta role as shown in Figure 3 is matched onto the crossing h-edge in the model. The square and diamond nodes indicate where the middle and end parts of each meta role in the query match. Crossings of v-edges are similar.

We now define counting query parts in a more precise way. Note that each counting query consists of $4n + 4$ meta roles. In the subsequent definition, $q_m^{i,j}$ is a meta role used in the counting query for A_i, where j ranges over $0, \dots, 4n + 3$.

Definition 1. *For all i, j with $1 \leq i \leq n$ and $0 \leq j < 4n+4$, put*

$$\begin{aligned} q_m^{i,j} := \{\ & r(v_1^{i,j}, v_0^{i,j}), v(v_1^{i,j}, v_2^{i,j}), r(v_3^{i,j}, v_2^{i,j}), v(v_4^{i,j}, v_3^{i,j}), r(v_5^{i,j}, v_4^{i,j}), h(v_5^{i,j}, v_6^{i,j}), \\ & r(v_7^{i,j}, v_6^{i,j}), h(v_8^{i,j}, v_7^{i,j}), r(v_9^{i,j}, v_8^{i,j}), r(v_9^{i,j}, v_{10}^{i,j}), h(v_{10}^{i,j}, v_{11}^{i,j}), r(v_{11}^{i,j}, v_{12}^{i,j}), \\ & h(v_{12}^{i,j}, v_{13}^{i,j}), r(v_{13}^{i,j}, v_{14}^{i,j}), v(v_{14}^{i,j}, v_{15}^{i,j}), r(v_{15}^{i,j}, v_{16}^{i,j}), v(v_{17}^{i,j}, v_{16}^{i,j}), r(v_{17}^{i,j}, v_0^{i,j+1})\} \end{aligned}$$

with $v_0^{i,4n+4} = v_0^{i,0}$. For each i with $1 \leq i \leq n$, the counting query for A_i *is*

$$q_c^i := \{\ A_i(v_0^{i,0}), \neg A_i(v_0^{i,1}), A_i(v_0^{i,2n+2}), \neg A_i(v_0^{i,2n+3})\} \ \cup \bigcup_{0 \leq j < 4n+4} q_m^{i,j}$$

with $v_9^{i,0} = v$, $v_9^{i,2n+2} = v'$. The counting query q_c for the whole counter is

$$q_c := \{B(v), Q(v), W(v'), Q(v')\} \cup \bigcup_{1 \leq i \leq n} q_c^i$$

Note that each counting query q_c^i is a cycle as intended since $v_0^{i,4n+4} = v_0^{i,0}$.

As explained above, the overall counting query q_c links a Q-node x_1 of a tree to the Q-nodes x_2 of its successor trees that represent the same bit position for the horizontal and vertical counters. To establish the central property $(*)$ in models of $\mathcal{K}_0$ that do *not* match the query q_0 to be constructed, it thus remains to modify q_0 such that it matches only if the L_n-predecessor x_1' of x_1 has the same truth assignment of X' and Y' as the L_n-predecessor x_2' of x_2 for X and Y. This is achieved by the second type of component queries in q_0, the copying components.

To prepare for these components, we introduce additional concept names $Q_0, \dots, Q_3$ that realize a unary counter of trees, counting modulo 4. These concept names will be used to identify the successor relation between the trees in query matches. The counter value is incremented when moving from a tree to a successor tree and propagated to all L_n- and L_{n+1}-nodes inside each tree and the ends of their side chains. This is achieved by adding to $\mathcal{K}_0$ the following concept inclusions:

$$\{o\} \sqsubseteq Q_0 \tag{31}$$

$$\begin{aligned} Q_i \sqcap B \sqcap L_0 \sqsubseteq\ & \forall h.\forall r.Q_{i+1 \bmod 4} \sqcap \forall h.Q_{i+2 \bmod 4} \sqcap \\ & \forall v.\forall r.Q_{i+1 \bmod 4} \sqcap \forall v.Q_{i+2 \bmod 4} \end{aligned} \qquad i=0,2 \tag{32}$$

$$Q_i \sqcap L_0 \sqsubseteq \forall (r^-; r)^n.(L_n \to Q_i) \qquad 0 \leq i \leq 3 \tag{33}$$

$$Q_i \sqsubseteq \neg Q_j \qquad 0 \leq i < j \leq 3 \tag{34}$$

$$\begin{aligned} Q_i \sqcap (L_n \sqcup L_{n+1}) \sqsubseteq\ & \forall h.\forall r.\forall h^-.\forall r.\forall v.\forall r.\forall v^-.\forall r.Q_i \sqcap \forall v.\forall r.\forall v^-.\forall r.Q_i \sqcap \\ & \forall h^-.\forall r.\forall v.\forall r.\forall v^-.\forall r.Q_i \sqcap \forall v^-.\forall r.Q_i \end{aligned} \qquad 0 \leq i \leq 3 \tag{35}$$

The copying components take the form displayed in Figure 7, i.e., there are 16 such components in total. Each component is like the upper half of a counting component, except that the concept labels have changed to negated conjunctions. In Figure 7, the four copying components in each row take care of one possible truth assignment to X' and Y' and the corresponding assignment to X and Y. We need four queries per truth assignment to deal with the four possible ways in which a counting query can match, as induced by the concept names $Q_0, \dots, Q_3$:

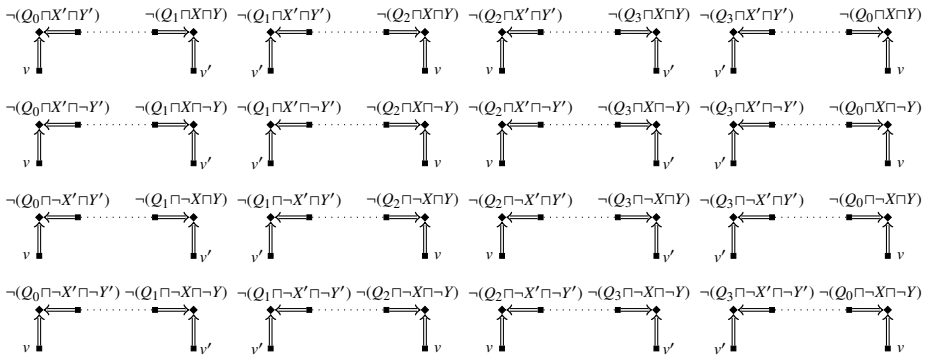

Fig. 7. The 16 query copying components

(a) v matches into a tree that satisfies B and Q_0, and v' into a successor tree that satisfies W and Q_1;
(b) v' matches into a tree that satisfies W and Q_1, and v into a successor tree that satisfies B and Q_2;
(c) v matches into a tree that satisfies B and Q_2, and v' into a successor tree that satisfies W and Q_3;
(d) v' matches into a tree that satisfies W and Q_3, and v into a successor tree that satisfies B and Q_0.

To explain in detail how the copying queries work, consider case (a). For simplicity, in the explanation we largely ignore side chains and pretend that meta roles are compositions $r; r^-$, as in our initial, simplified view on counting components. Take the Q-node x_1 of a tree that satisfies B and Q_0 and the Q-node x_2 of a successor tree that satisfies W and Q_1. The relevant queries are those from the leftmost column in Figure 7. Let x_i' be the predecessor L_n-node of x_i, and x_i'' the successor L_{n+1}-node of x_i. Due to the Q-label in the *counting* queries, the variable v can only be matched to x_1 and the variable v' can only be matched to x_2. Let u be the neighboring variable of v in the copying components, and u' the neighboring variable of v'. Then u can only be matched to either x_1' or x_1'' while u' can only be matched to either x_2' or x_2''. The match $u \mapsto x_1''$ and $u' \mapsto x_2''$ is excluded because the path from u to u' is not long enough. So a component has a match iff either x_1' satisfies the label of u or x_2' satisfies the label of u' (since x_1' and x_2'' have complementary truth values for X' and Y', one of them always satisfy the label of u and similarly for x_1'', x_2'' and u'). The four copying components in the first column exclude exactly four situations when x_1' has the same truth assignment of X' and Y' as x_2' for X and Y. It remains to note that the copying components in the other columns always have a match in this situation due to our use of the concept names $Q_0, \dots, Q_3$ in the labels, and thus do not interfere with the matches of the queries in the leftmost components.

We remark that, without the use of the concept names $Q_0, \dots, Q_3$, it does not seem possible to ensure proper directionality of copying. For example, copying components that copy the X/Y-assignment from a black tree to the X'/Y'-assignment in white successor trees would also copy this assignment to the the X'/Y'-assignment in white *predecessor* trees. A formal definition of copying queries can be found in [9].

This finishes the construction of the query q_0 and of the part of $\mathcal{K}_0$ that enforces the torus structure. It remains to encode tilings of the domino system D_0.

$$\top \sqsubseteq T_1 \sqcup \cdots \sqcup T_k \qquad T_i \sqcap T_j \sqsubseteq \bot \qquad 1 \leq i < j \leq k \tag{36}$$

$$T_i \sqcap \exists h.T_j \sqsubseteq \bot \qquad T_k \sqcap \exists v.T_\ell \sqsubseteq \bot \qquad \langle i, j\rangle \notin H, \langle k, \ell\rangle \notin V \tag{37}$$

Finally, we enforce the initial condition $c^0 = \langle t_1^0, \ldots, t_n^0\rangle$ of the torus.

$$\{o\} \sqsubseteq T_{t_1^0} \sqcap \forall h.(T_{t_2^0} \sqcap \forall h.(T_{t_3^0} \sqcap \forall h.(T_{t_4^0} \sqcap \ldots \forall h.T_{t_n^0} \ldots))) \tag{38}$$

More details regarding the correctness of the reduction can be found in [9]. The most challenging issue is to show that when D_0 admits a tiling with initial condition c^0 and we build a model $\mathcal{I}$ of $\mathcal{K}$ that has the intended torus shape, then $\mathcal{I} \not\models q_0$: we need to prove that there are no unintended foldings and matchings of the query q_0.

Theorem 1. *Conjunctive query entailment by $\mathcal{ALCOIF}$ knowledge bases is* co-N2ExpTime*-hard.*

Acknowledgments B. Glimm is supported by EPSRC grant EP/F065841/1, Y. Kazakov by EPSRC grant EP/G02085X, and C. Lutz by DFG SFB/TR 8 "Spatial Cognition".

References

1. Rudolph, S., Glimm, B.: Nominals, inverses, counting, and conjunctive queries. J. of Artificial Intelligence Research **39** (2010) 429–481
2. Lutz, C.: Inverse roles make conjunctive queries hard. In: Proc. of the 2007 Description Logic Workshop (DL 2007). (2007)
3. Lutz, C.: The complexity of conjunctive query answering in expressive description logics. In: Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR-08), LNCS (2008) 179–193
4. Glimm, B., Horrocks, I., Lutz, C., Sattler, U.: Conjunctive query answering for the description logic $\mathcal{SHIQ}$. J. of Artificial Intelligence Research **31** (2008) 151–198
5. Glimm, B., Horrocks, I., Sattler, U.: Unions of conjunctive queries in $\mathcal{SHOQ}$. In: Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-08), AAAI Press/The MIT Press (2008)
6. Calvanese, D., Eiter, T., Ortiz, M.: Regular path queries in expressive description logics with nominals. In: Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI-09). (2009) 714–720
7. Kazakov, Y.: $\mathcal{RIQ}$ and $\mathcal{SROIQ}$ are harder than $\mathcal{SHOIQ}$. In: Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-08), AAAI Press/The MIT Press (2008)
8. Glimm, B., Kazakov, Y.: Role conjunctions in expressive description logics. In: Proc. of the 15th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR 2008). Volume 5330 of LNCS., Springer (2008) 391–405
9. Glimm, B., Kazakov, Y., Lutz, C.: Status $\mathcal{QIO}$: An update. Technical report, The University of Oxford (2011) `http://www.comlab.ox.ac.uk/files/3969/GlKL11a.pdf`.
10. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: The Description Logic Handbook. Cambridge University Press (2003)
11. Börger, E., Grädel, E., Gurevich, Y.: The Classical Decision Problem. Perspectives in Mathematical Logic. Springer (1997) Second printing, Springer Verlag, 2001.
12. Tobies, S.: The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. J. of Artificial Intelligence Research **12** (2000)