Containment in Monadic Disjunctive Datalog, MMSNP, and Expressive Description Logics

Pierre Bourhis
CNRS CRIStAL, UMR 8189, Université Lille 1,
INRIA Lille, France
pierre.bourhis@univ-lille1.fr

Carsten Lutz
University of Bremen, Germany
clu@uni-bremen.de

Abstract

We study query containment in three closely related formalisms: monadic disjunctive Datalog (MDDLog), MMSNP (a logical generalization of constraint satisfaction problems), and ontology-mediated queries (OMQs) based on expressive description logics and unions of conjunctive queries. Containment in MMSNP was known to be decidable due to a result by Feder and Vardi, but its exact complexity has remained open. We prove 2NExpTime-completeness and extend this result to monadic disjunctive Datalog and to OMQs.

Introduction

In knowledge representation with ontologies, data centric applications have become a significant subject of research. In such applications, ontologies are used to address incompleteness and heterogeneity of the data, and for enriching it with background knowledge (Calvanese et al. 2009). This trend has given rise to the notion of an *ontology-mediated query (OMQ)* which combines a database query with an ontology, often formulated in a description logic (DL). From a data centric viewpoint, an OMQ can be viewed as a normal database query that happens to consist of two components (the ontology and the actual query). It is thus natural to study OMQs in the same way as other query languages, aiming to understand e.g. their expressive power and the complexity of fundamental reasoning tasks such as query containment. In this paper, we concentrate on the latter.

Containment of OMQs was first studied in (Levy and Rousset 1996; Calvanese, De Giacomo, and Lenzerini 1998) and more recently in (Calvanese, Ortiz, and Simkus 2011; Bienvenu et al. 2014; Bienvenu, Lutz, and Wolter 2012). To appreciate the usefulness of this reasoning task, it is important to recall that real-world ontologies can be very large and tend to change frequently. As a user of OMQs, one might thus want to know whether the ontology used in an OMQ can be replaced with a potentially much smaller module extracted from a large ontology or with a newly released version of the ontology, without compromising query answers. This requires to decide equivalence of OMQs, which can be done by answering two containment questions. Containment can also serve as a central reasoning service when optimizing OMQs in static analysis (Bienvenu, Lutz, and Wolter 2012).

In the most general form of OMQ containment, the two OMQs can involve different ontologies and the data schema (ABox signature, in DL terms) can be restricted to a subset of the signature of the ontologies. While results for this form of containment have been obtained for inexpressive DLs such as those of the DL-Lite and $\mathcal{EL}$ families (Bienvenu, Lutz, and Wolter 2012), containment of OMQs based on expressive DLs turned out to be a technically challenging problem. A step forward has been made in (Bienvenu et al. 2014) where it was observed that there is a close relationship between three groups of formalisms: (i) OMQs based on expressive DLs, (ii) monadic disjunctive Datalog (MDDLog) programs, and (iii) constraint satisfaction problems (CSPs) as well as their logical generalization MMSNP. These observations have given rise to first complexity results for containment of OMQs based on expressive DLs, namely NExpTime-completeness for several cases where the actual query is an atomic query of the form $A(x)$, with A a monadic relation.

In this paper, we study containment in MDDLog, MMSNP, and OMQs that are based on expressive DLs, conjunctive queries (CQ), and unions thereof (UCQs). A relevant result is due to Feder and Vardi (1998) who show that containment of MMSNP sentences is decidable and that this gives rise to decidability results for CSPs such as whether the complement of a CSP is definable in monadic Datalog. As shown in (Bienvenu et al. 2014), the complement of MMSNP is equivalent to Boolean MDDLog programs and to Boolean OMQs with UCQs as the actual query. While these results can be used to infer decidability of containment in the mentioned query languages, they do not immediately yield tight complexity bounds. In particular, Feder and Vardi describe their algorithm for containment in MMSNP only on a very high level of abstraction, do not analyze its complexity, and do not attempt to provide lower bounds. Also a subsequent study of MMSNP containment and related problems did not clarify the precise complexity (Madelaine 2010). Other issues to be addressed are that MMSNP containment corresponds only to the containment of *Boolean* queries and that the translation of OMQs into MMSNP involves a double exponential blowup.

Our main contribution is to show that all of the mentioned containment problems are 2NEXPTIME-complete. In particular, this is the case for MDDLog, MMSNP, OMQs whose ontology is formulated in a DL between $\mathcal{ALC}$ and $\mathcal{SHI}$ and where the actual queries are UCQs, and OMQs whose ontology is formulated in a DL between $\mathcal{ALCI}$ and $\mathcal{SHI}$ and where the actual queries are CQs. This closes open problems from (Madelaine 2010) about MMSNP containment and from (Bienvenu, Lutz, and Wolter 2012) about OMQ containment. In addition, clarifying the complexity of MDDLog containment is interesting from the perspective of database theory, where Datalog containment has received a lot of attention. While being undecidable in general (Shmueli 1993), containment is known to be decidable for monadic Datalog (Cosmadakis et al. 1988) and is in fact 2EXPTIME-complete (Benedikt, Bourhis, and Senellart 2012). Here, we show that adding disjunction increases the complexity to 2NEXPTIME. We refer to (Bourhis, Krötzsch, and Rudolph 2015) for another recent work that generalizes monadic Datalog containment, in an orthogonal direction. It is interesting to note that all these previous works rely on the existence of witness instances for non-containment which have a tree-like shape. In MDDLog containment, such witnesses are not guaranteed to exist which results in significant technical challenges.

This paper is structured as follows. We first concentrate on MDDLog containment, establishing that containment of a Boolean MDDLog program in a Boolean conjunctive query (CQ) is 2NEXPTIME-hard and that containment in Boolean MDDLog is in 2NEXPTIME. We then generalize the upper bound to programs that are non-Boolean and admit constant symbols. The lower bound uses a reduction of a tiling problem and borrows queries from (Björklund, Martens, and Schwentick 2008). For the upper bound, we essentially follow the arguments of Feder and Vardi (1998), but provide full details of all involved constructions and carefully analyze the involved blowups. It turns out that an MDDLog containment question $\Pi_1 \subseteq \Pi_2$ can be decided non-deterministically in time single exponential in the size of Π_1 and double exponential in the size of Π_2. Together with some straightforward observations, this also settles the complexity of MMSNP containment. We additionally observe that FO- and Datalog-rewritability of MDDLog programs and (the complements of) MMSNP sentences is 2NEXPTIME-hard.

We then consider containment between OMQs, starting with the observation that the 2NEXPTIME lower bound for MDDLog also yields that containment of an OMQ in a CQ is 2NEXPTIME-hard when ontologies are formulated in $\mathcal{ALC}$ and UCQs are used as queries. The same is true for the containment of an OMQ in an OMQ, even when their ontologies are identical. We then establish a matching upper bound by translating OMQs to MDDLog and applying our results for MDDLog containment. It is interesting that the complexity is double exponential only in the size of the actual query (which tends to be very small) and only single exponential in the size of the ontology. We finally establish another 2NEXPTIME lower bound which applies to containment of OMQs whose ontologies are formulated in $\mathcal{ALCI}$ and whose actual queries are CQs (instead of UCQs as in the first lower bound). This requires a different reduction strategy which borrows queries from (Lutz 2008).

Due to space limitations, we defer proof details to the appendix, available at http://www.informatik.uni-bremen.de/tdki/research/papers.html.

Preliminaries

A *schema* is a finite collection $\mathbf{S} = (S_1, \ldots, S_k)$ of relation symbols with associated non-negative arity. An $\mathbf{S}$-*fact* is an expression of the form $S(a_1, \ldots, a_n)$ where $S \in \mathbf{S}$ is an n-ary relation symbol, and $a_1, \ldots, a_n$ are elements of some fixed, countably infinite set const of *constants*. An $\mathbf{S}$-*instance* I is a finite set of $\mathbf{S}$-facts. The *active domain* $\mathsf{adom}(I)$ of I is the set of all constants that occur in the facts in I.

An $\mathbf{S}$-*query* is semantically defined as a mapping q that associates with every $\mathbf{S}$-instance I a set of *answers* $q(I) \subseteq \mathsf{adom}(I)^n$, where $n \geq 0$ is the *arity* of q. If $n = 0$, then we say that q is *Boolean* and we write $I \models q$ if $() \in q(I)$. We now introduce some concrete query languages. A *conjunctive query (CQ)* takes the form $\exists \mathbf{y}\, \varphi(\mathbf{x}, \mathbf{y})$ where φ is a conjunction of relational atoms and $\mathbf{x}$, $\mathbf{y}$ denote tuples of variables. The variables in $\mathbf{x}$ are called *answer variables*. Semantically, $\exists \mathbf{y}\, \varphi(\mathbf{x}, \mathbf{y})$ denotes the query

$$q(I) = \{(a_1, \ldots, a_n) \in \mathsf{adom}(I)^n \mid I \models \varphi[a_1, \ldots, a_n]\}.$$

A *union of conjunctive queries (UCQ)* is a disjunction of CQs with the same free variables. We now define disjunctive Datalog programs, see also (Eiter, Gottlob, and Mannila 1997). A *disjunctive Datalog rule* ρ has the form

$$S_1(\mathbf{x}_1) \vee \cdots \vee S_m(\mathbf{x}_m) \leftarrow R_1(\mathbf{y}_1) \wedge \cdots \wedge R_n(\mathbf{y}_n)$$

where $n > 0$ and $m \geq 0$.[1] We refer to $S_1(\mathbf{x}_1) \vee \cdots \vee S_m(\mathbf{x}_m)$ as the *head* of ρ, and to $R_1(\mathbf{y}_1) \wedge \cdots \wedge R_n(\mathbf{y}_n)$ as the *body*. Every variable that occurs in the head of a rule ρ is required to also occur in the body of ρ. A *disjunctive Datalog (DDLog) program* Π is a finite set of disjunctive Datalog rules with a selected *goal relation* goal that does not occur in rule bodies and appears only in non-disjunctive *goal rules* $\mathsf{goal}(\mathbf{x}) \leftarrow R_1(\mathbf{x}_1) \wedge \cdots \wedge R_n(\mathbf{x}_n)$. The *arity of* Π is the arity of the goal relation. Relation symbols that occur in the head of at least one rule of Π are *intensional (IDB) relations*, and all remaining relation symbols in Π are *extensional (EDB) relations*. Note that, by definition, goal is an IDB relation. A DDLog program is called *monadic* or an *MDDLog program* if all its IDB relations except goal have arity at most one.

An $\mathbf{S}$-instance, with $\mathbf{S}$ the set of all (IDB and EDB) relations in Π, is a *model* of Π if it satisfies all rules in Π. We use $\mathsf{Mod}(\Pi)$ to denote the set of all models of Π. Semantically, a DDLog program Π of arity n defines the following query over the schema $\mathbf{S}_E$ that consists of the EDB relations of Π: for every $\mathbf{S}_E$-instance I,

$$\Pi(I) = \{\mathbf{a} \in \mathsf{adom}(I)^n \mid \mathsf{goal}(\mathbf{a}) \in J \text{ for all } J \in \mathsf{Mod}(\Pi) \text{ with } I \subseteq J\}.$$

[1]Empty rule heads (denoted $\bot$) are sometimes disallowed. We admit them only in our upper bound proofs, but do not use them for lower bounds, thus achieving maximum generality.

Let Π_1, Π_2 be DDLog programs over the same EDB schema $\mathbf{S}_E$ and of the same arity. We say that Π_1 *is contained in* Π_2, written $\Pi_1 \subseteq \Pi_2$, if for every $\mathbf{S}_E$-instance I, we have $\Pi_1(I) \subseteq \Pi_2(I)$.

Example 1. *Consider the following MDDLog program Π_1 over EDB schema $\mathbf{S}_E = \{A, B, r\}$:*

$$\begin{aligned} A_1(x) \vee A_2(x) &\leftarrow A(x) \\ \mathsf{goal}(x) &\leftarrow A_1(x) \wedge r(x,y) \wedge A_1(y) \\ \mathsf{goal}(x) &\leftarrow A_2(x) \wedge r(x,y) \wedge A_2(y) \end{aligned}$$

Let Π_2 consist of the single rule $\mathsf{goal}(x) \leftarrow B(x)$. Then $\Pi_1 \not\subseteq \Pi_2$ is witnessed, for example, by the $\mathbf{S}_E$-instance $I = \{r(a,a), A(a)\}$. It is interesting to note that there is no tree-shaped $\mathbf{S}_E$-instance that can serve as a witness although all rule bodies in Π_1 and Π_2 are tree-shaped. In fact, a tree-shaped instance does not admit any answers to Π_1 because we can alternate A_1 and A_2 with the levels of the tree, avoiding to make goal *true anywhere.*

An *MMSNP sentence* over schema $\mathbf{S}_E$ has the form $\exists X_1 \cdots \exists X_n \forall x_1 \cdots \forall x_m \varphi$ with $X_1, \dots, X_n$ monadic second-order variables, $x_1, \dots, x_m$ first-order variables, and φ a conjunction of formulas of the form

$$\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \beta_1 \vee \cdots \vee \beta_m \text{ with } n, m \geq 0,$$

where each α_i takes the form $X_i(x_j)$ or $R(\mathbf{x})$ with $R \in \mathbf{S}_E$, and each β_i takes the form $X_i(x_j)$. This presentation is syntactically different from, but semantically equivalent to the original definition from (Feder and Vardi 1998), which does not use the implication symbol and instead restricts the allowed polarities of atoms. An MMSNP sentence φ can serve as a Boolean query in the obvious way, that is, $I \models \varphi$ whenever φ evaluates to true on the instance I. The containment problem in MMSNP coincides with logical implication. See (Bodirsky, Chen, and Feder 2012; Bodirsky and Dalmau 2013) for more information on MMSNP.

It was shown in (Bienvenu et al. 2014) that the complement of an MMSNP sentence can be translated into an equivalent Boolean MDDLog program in polynomial time and vice versa. The involved complementation is irrelevant for the purposes of deciding containment since for any two Boolean queries q_1, q_2, we have $q_1 \subseteq q_2$ if and only if $\neg q_1 \not\supseteq \neg q_2$. Consequently, any upper bound for containment in MDDLog also applies to MMSNP and so does any lower bound for containment between Boolean MDDLog programs.

Example 2. *Let $\mathbf{S}_E = \{r\}$, r binary. The complement of the MMSNP formula $\exists R \exists G \exists B \forall x \forall y\, \psi$ over $\mathbf{S}_E$ with ψ the conjunction of*

$$\begin{aligned} \top &\rightarrow R(x) \vee G(x) \vee B(x) \\ C(x) \wedge r(x,y) \wedge C(y) &\rightarrow \bot \quad \textit{for } C \in \{R,G,B\} \end{aligned}$$

is equivalent to the Boolean MDDLog program

$$\begin{aligned} r(x,y) &\rightarrow C(x) \vee \overline{C}(x) \quad \textit{for } C \in \{R,G,B\} \\ r(x,y) &\rightarrow C(y) \vee \overline{C}(y) \quad \textit{for } C \in \{R,G,B\} \\ \overline{R}(x) \wedge \overline{G}(x) \wedge \overline{B}(x) &\rightarrow \mathsf{goal}() \\ C(x) \wedge r(x,y) \wedge C(y) &\rightarrow \mathsf{goal}() \quad \textit{for } C \in \{R,G,B\}. \end{aligned}$$

MDDLog and MMSNP: Lower Bounds

The first main aim of this paper is to establish the following result. Point 3 closes an open problem from (Madelaine 2010).

Theorem 3. *The following containment problems are* 2NExpTime*-complete:*

1. *of an MDDLog program in a CQ;*
2. *of an MDDLog program in an MDDLog program;*
3. *of two MMSNP sentences.*

We prove the lower bounds by reduction of a tiling problem. It suffices to show that containment between a Boolean MDDLog program and a Boolean CQ is 2NExpTime-hard. A *2-exp square tiling problem* is a triple $P = (\mathbb{T}, \mathbb{H}, \mathbb{V})$ where

- $\mathbb{T} = \{T_1, \dots, T_p\}$, $p \geq 1$, is a finite set of *tile types*;
- $\mathbb{H} \subseteq \mathbb{T} \times \mathbb{T}$ is a *horizontal matching relation*;
- $\mathbb{V} \subseteq \mathbb{T} \times \mathbb{T}$ is a *vertical matching relation*.

An input to P is a word $w \in \mathbb{T}^*$. Let $w = T_{i_0} \cdots T_{i_n}$. A *tiling for P and w* is a map $f : \{0, \dots, 2^{2^n} - 1\} \times \{0, \dots, 2^{2^n} - 1\} \rightarrow \mathbb{T}$ such that $f(0,j) = T_{i_j}$ for $0 \leq j \leq n$, $(f(i,j), f(i+1,j)) \in \mathbb{H}$ for $0 \leq i < 2^{2^n}$, and $(f(i,j), f(i,j+1)) \in \mathbb{V}$ for $0 \leq i < 2^{2^n}$. It is 2NExpTime-hard to decide, given a 2-exp square tiling problem P and an input w to P, whether there is a tiling for P and w.

For the reduction, let P be a 2-exp square tiling problem and w_0 an input to P of length n. We construct a Boolean MDDLog program Π and a Boolean CQ q such that $\Pi \subseteq q$ iff there is a tiling for P and w_0. To get a first intuition, assume that instances I have the form of a (potentially partial) $2^{2^n} \times 2^{2^n}$-grid in which the horizontal and vertical positions of grid nodes are identified by binary counters, described in more detail later on. We construct q such that $I \models q$ iff I contains a counting defect, that is, if the counters in I are not properly incremented or assign multiple counter values to the same node. Π is constructed such that on instances I without counting defects, $I \models \Pi$ iff the partial grid in I does not admit a tiling for P and w_0. Note that this gives the desired result: if there is no tiling for P and w_0, then an instance I that represents the full $2^{2^n} \times 2^{2^n}$-grid (without counting defects) shows $\Pi \not\subseteq q$; conversely, if there is a tiling for P and w_0, then $I \not\models q$ means that there is no counting defect in I and thus $I \not\models \Pi$.

We now detail the exact form of the grid and the counters. Some of the constants in the input instance serve as *grid nodes* in the $2^{2^n} \times 2^{2^n}$-grid while other constants serve different purposes described below. To identify the position of a grid node a, we use a binary counter whose value is stored at the 2^m leaves of a binary *counting tree* with root a and depth $m := n + 1$. The depth of counting trees is m instead of n because we need to store the horizontal position (first 2^n bits of the counter) as well as the vertical position (second 2^n bits). The binary relation r is used to connect successors. To distinguish left and right successors, every left successor a has an attached *left*

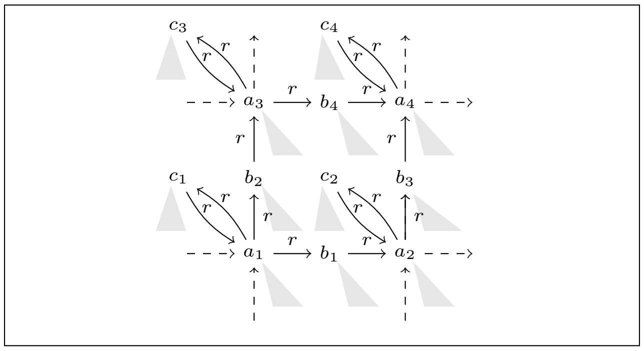


Figure 1: A grid cell (counting trees in grey).

navigation gadget $r(a, a_1), r(a_1, a_2), \mathsf{jump}(a, a_2)$ and every right successor a has an attached *right navigation gadget* $r(a, a_1), \mathsf{jump}(a, a_1)$—these gadgets will be used in the formulation of the query q later on.

If a grid node a_2 represents the right neighbor of grid node a_1, then there is some node b such that $r(a_1, b), r(b, a_2)$. The node b is called a *horizontal step node*. Likewise, if a_2 represents the upper neighbor of a_1, then there must also be some b with $r(a_1, b), r(b, a_2)$ and we call b a *vertical step node*. In addition, for each grid node a there must be a node c such that $r(a, c), r(c, a)$ and we call c a *self step node*. We make sure that, just like grid nodes, all three types of step node have an attached counting tree. Figure 1 illustrates the representation of a single grid cell.

We need to make sure that counters are properly incremented when transitioning to right and upper neighbors via step nodes. To achieve this, each counting tree actually stores two counter values via monadic relations $B_1, \overline{B}_1$ (first value) and $B_2, \overline{B_2}$ (second value) at the leaves of the tree, where B_i indicates bit value one and $\overline{B}_i$ bit value zero. While the B_1-value represents the actual position of the node in the grid, the B_2-value is copied from the B_1-values of its predecessor nodes (which must be identical). In fact, the query q to be defined later shall guarantee that

(Q1) whenever $r(a_1, a_2)$ and a_1 is associated (via a counting tree) with B_1-value k_1 and a_2 is associated with B_2-value k_2, then $k_1 = k_2$;

(Q2) every node is associated (via counting trees) with at most one B_1-value.

Between neighboring grid and step nodes, counter values are thus copied as described in (Q1) above, but not incremented. Incrementation takes place inside counting trees, as follows: at grid nodes and at self step nodes, the two values are identical; at horizontal (resp. vertical) step nodes, the B_1-value is obtained from the B_2-value by incrementing the horizontal part and keeping the vertical part (resp. incrementing the vertical part and keeping the horizontal part).

We now construct the program Π. As the EDB schema, we use $\mathbf{S}_E = \{r, \mathsf{jump}, B_1, B_2, \overline{B}_1, \overline{B_2}\}$ where r and jump are binary and all other relations are monadic. We first define rules which verify that a grid or self step node has a proper counting tree attached to it (in which both counters are identical):

$$\begin{aligned}
\mathsf{left}(x) &\leftarrow r(x, y) \wedge r(y, z) \wedge \mathsf{jump}(x, z) \\
\mathsf{right}(x) &\leftarrow r(x, y) \wedge \mathsf{jump}(x, y) \\
\mathsf{lrok}(x) &\leftarrow \mathsf{left}(x) \\
\mathsf{lrok}(x) &\leftarrow \mathsf{right}(x) \\
\mathsf{lev}_m^G(x) &\leftarrow B_1(x) \wedge B_2(x) \wedge \mathsf{lrok}(x) \\
\mathsf{lev}_m^G(x) &\leftarrow \overline{B}_1(x) \wedge \overline{B}_2(x) \wedge \mathsf{lrok}(x) \\
\mathsf{lev}_i^G(x) &\leftarrow r(x, y_1) \wedge \mathsf{lev}_{i+1}^G(y_1) \wedge \mathsf{left}(y_1) \wedge \\
&\qquad r(x, y_2) \wedge \mathsf{lev}_{i+1}^G(y_2) \wedge \mathsf{right}(y_2)
\end{aligned}$$

for $0 \leq i < m$. We call a constant a of an instance I *g-active* if it has all required structures attached to serve as a grid node. Such constants are marked by the IDB relation gactive:

$$\mathsf{gactive}(x) \leftarrow \mathsf{lev}_0^G(x) \wedge r(x, y) \wedge \mathsf{lev}_0^G(y) \wedge r(y, x)$$

We also want horizontal and vertical step nodes to be roots of the required counting trees. The difference to the counting trees below grid / self step nodes is that we need to increment the counters. This requires modifying the rules with head relation lev_i^G above. We only consider horizontal step nodes explicitly as vertical ones are very similar. The relations $B_1, \overline{B_1}$ and $B_2, \overline{B_2}$ give rise to a labeling of the leaf nodes that defines a word over the alphabet $\Sigma = \{0, 1\}^2$ where symbol (i, j) means that the bit encoded via $B_1, \overline{B}_1$ has value i and the bit encoded via $B_2, \overline{B}_2$ has value j. Ensuring that the B_1-value is obtained by incrementation from the B_2-value (least significant bit at the left-most leaf) then corresponds to enforcing that the leaf word is from the regular language $L = (0, 1)^*(1, 0)((0, 0) + (1, 1))^*$. To achieve this, we consider the languages $L_1 = (0, 1)^*$, $L_2 = L$, and $L_3 = ((0, 0) + (1, 1))^*$. Instead of level relations lev_i^G, we use relations $\mathsf{lev}_i^{H,\ell}$ where $\ell \in \{1, 2, 3\}$ indicates that the leaf word of the subtree belongs to the language L_ℓ:

$$\begin{aligned}
\mathsf{lev}_m^{H,1}(x) &\leftarrow \overline{B}_1(x) \wedge B_2(x) \wedge \mathsf{lrok}(x) \\
\mathsf{lev}_m^{H,2}(x) &\leftarrow B_1(x) \wedge \overline{B}_2(x) \wedge \mathsf{lrok}(x) \\
\mathsf{lev}_m^{H,3}(x) &\leftarrow B_1(x) \wedge B_2(x) \wedge \mathsf{lrok}(x) \\
\mathsf{lev}_m^{H,3}(x) &\leftarrow \overline{B}_1(x) \wedge \overline{B}_2(x) \wedge \mathsf{lrok}(x) \\
\mathsf{lev}_i^{H,\ell_3}(x) &\leftarrow r(x, y_1) \wedge \mathsf{lev}_{i+1}^{H,\ell_1}(y_1) \wedge \mathsf{left}(y_1) \wedge \\
&\qquad r(x, y_2) \wedge \mathsf{lev}_{i+1}^{H,\ell_2}(y_2) \wedge \mathsf{right}(y_2)
\end{aligned}$$

where $1 \leq i < m$ and $(\ell_1, \ell_2, \ell_3) \in \{(1, 1, 1), (1, 2, 2), (2, 3, 2), (3, 3, 3)\}$. We call a constant of an instance *h-active* if it is the root of a counting tree that implements incrementation of the horizontal position (left subtree of the root) and does not change the vertical position (right subtree of the root), identified by the IDB relation hactive:

$$\begin{aligned}
\mathsf{hactive}(x) &\leftarrow r(x, y_1) \wedge \mathsf{lev}_1^{H,\ell_2}(y_1) \wedge \mathsf{left}(y_1) \wedge \\
&\qquad r(x, y_2) \wedge \mathsf{lev}_1^{H,\ell_3}(y_2) \wedge \mathsf{right}(y_2)
\end{aligned}$$

We omit the rules for the corresponding IDB relation vactive. Call the fragment of Π that we have constructed up to this point Π_{tree}.

Recall that we want an instance to make Π true if it admits no tiling for P and w. We thus label all g-active nodes with a tile type:

$$\bigvee_{T_i \in \mathbb{T}} T_i(x) \leftarrow \mathsf{gactive}(x)$$

It then remains to trigger the goal relation whenever there is a defect in the tiling. Thus add for all $T_i, T_j \in \mathbb{T}$ with $(T_i, T_j) \notin H$:

$$\mathsf{goal}() \leftarrow T_i(x) \wedge \mathsf{gactive}(x) \wedge r(x,y) \wedge \mathsf{hactive}(y) \wedge r(y,z) \wedge T_j(z) \wedge \mathsf{gactive}(z)$$

and for all $T_i, T_j \in \mathbb{T}$ with $(T_i, T_j) \notin V$:

$$\mathsf{goal}() \leftarrow T_i(x) \wedge \mathsf{gactive}(x) \wedge r(x,y) \wedge \mathsf{vactive}(y) \wedge r(y,z) \wedge T_j(z) \wedge \mathsf{gactive}(z)$$

The last kind of defect concerns the initial condition. Let $w_0 = T_{i_0} \cdots T_{i_{n-1}}$. It is tedious but not difficult to write rules which ensure that, for all $i < n$, every g-active element whose B_1-value represents horizontal position i and vertical position 0 satisfies the monadic IDB relation $\mathsf{pos}_{i,0}$. We then put for all $j < n$ and all $T_\ell \in \mathbb{T}$ with $T_\ell \neq T_{i_j}$:

$$\mathsf{goal}() \leftarrow \mathsf{pos}_{j,0}(x) \wedge T_\ell(x).$$

We now turn to the definition of q; recall that we want it to achieve conditions (Q1) and (Q2) above. Due to the presence of self step nodes and since the counting trees below self step nodes and grid nodes must have identical values for the two counters, it can be verified that (Q1) implies (Q2). Therefore, we only need to achieve (Q1). We use as q a minor variation of a CQ constructed in (Björklund, Martens, and Schwentick 2008) for a similar purpose. We first constuct a UCQ and show in the appendix how to replace it with a CQ, which also involves some minor additions to the program Π_{tree} above.

The UCQ q makes essential use of the left and right navigation gadgets in counting trees. It uses a subquery $q_m(x, y)$ constructed such that x and y can only be mapped to corresponding leaves in successive counting trees, that is, (i) the roots of the trees are connected by the relation r and (ii) x can be reached from the root of the first tree by following the same sequence of left and right successors that one also needs to follow to reach y from the root of the second tree. To define $q_m(x, y)$, we inductively define queries $q_i(x, y)$ for all $i \leq m$, starting with $q_0(x, y) = r(x_0, y_0)$ and setting, for $0 < i \leq m$,

$$\begin{aligned} q_i(x_i,y_i) = {} & \exists x_{i-1} \exists y_{i-1} \exists z_{i,0} \cdots \exists z_{i,i+2} \exists z'_{i,1} \cdots \exists z'_{i,i+3} \\ & q_{i-1}(x_{i-1}, y_{i-1}) \wedge r(x_{i-1}, x_i) \wedge r(y_{i-1}, y_i) \wedge \\ & \mathsf{jump}(x_i, z_{i,i+2}) \wedge \mathsf{jump}(y_i, z'_{i,i+3}) \wedge \\ & r(z_{i,0}, z_{i,1}) \wedge \cdots \wedge r(z_{i,i+1}, z_{i,i+2}) \wedge \\ & r(z_{i,0}, z'_{i,1}) \wedge r(z_{i,1}, z'_{i,2}) \wedge \cdots \wedge r(z'_{i,i+2}, z'_{i,i+3}) \end{aligned}$$

The r-atom in q_0 corresponds to the move from the root of one counting tree to the root of a successive tree, the atoms $r(x_{i-1}, x_i)$ and $r(y_{i-1}, y_i)$ in q_i correspond to moving down the i-th step in both trees, and the remaining atoms in q_i make sure that both of these steps are to a left successor or to a right successor. We make essential use of the jump relation here, which shortcuts an edge on the path to the root for left successors, but not for right successors. Additional explanation is provided in the appendix. It is now easy to define the desired UCQ that achieves (Q1):

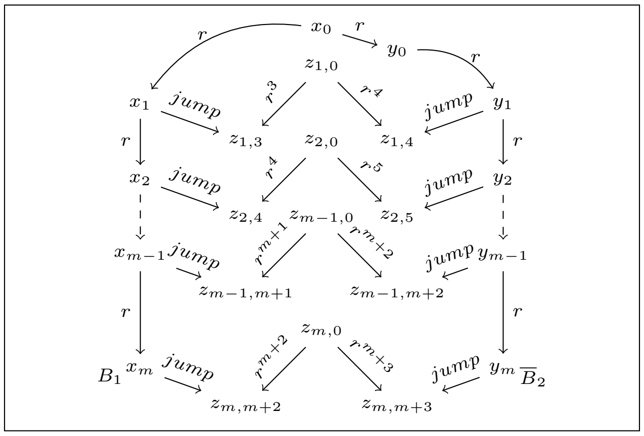


Figure 2: The first CQ in q.

$$\begin{aligned} q = {} & \exists x_m \exists y_m \, q_m(x_m, y_m) \wedge B_1(x_m) \wedge \overline{B}_2(y_m) \\ & \vee \exists x_m \exists y_m \, q_m(x_m, y_m) \wedge \overline{B}_1(x_m) \wedge B_2(y_m) \end{aligned}$$

The first CQ in q is displayed in Figure 4.

Lemma 4. $\Pi \not\subseteq q$ *iff there is no tiling for P and w_0.*

This finishes the proof of the lower bounds stated in Theorem 3. Before proceeding, we note that the lower bound can be adapted to important rewritability questions. A query is *FO-rewritable* if there is an equivalent first-order query and *(monadic) Datalog-rewritable* if there is an equivalent (non-disjunctive) (monadic) Datalog query. FO-Rewritability of a query is desirable since it allows to use conventional SQL database systems for query answering, and likewise for Datalog-rewritability and Datalog engines. For this reason, FO- and Datalog-rewritability have received a lot of attention. For example, they have been studied for OMQs in (Bienvenu et al. 2014) and for CSPs in (Feder and Vardi 1998; Larose, Loten, and Tardif 2007). Monadic Datalog is an interesting target as it constitutes an extremely well-behaved fragment of Datalog. It is open whether the known decidability of FO- and (monadic) Datalog-rewritability generalizes from CSPs to MMSNP. We observe here that these problems are at least 2NEXPTIME-hard. The proof is by a simple modification of the reduction presented above.

Theorem 5. *For MDDLog programs and the complements of MMSNP sentences, rewritability into FO, into monadic Datalog, and into Datalog are 2NEXPTIME-hard.*

MDDLog and MMSNP: Upper Bounds

The aim of this section is to establish the upper bounds stated in Theorem 3. It suffices to concentrate on MDDLog since the result for MMSNP follows. We first consider only Boolean MDDLog programs and then show how to extend the upper bound to MDDLog programs of any arity.

Our main algorithm is essentially the one described in (Feder and Vardi 1998). Since the constructions are described by Feder and Vardi only on an extremely high level of abstraction and without providing any analysis of the algorithm's running time, we give full details and proofs (in the appendix). The algorithm for deciding $\Pi_1 \subseteq \Pi_2$ proceeds in three steps. First, Π_1 and Π_2 are converted into a simplified form, then containment between the resulting programs Π_1^S and Π_2^S is reduced to a certain emptiness problem, and finally that problem is decided. A technical complication is posed by the fact that the construction of Π_1^S and Π_2^S does not preserve containment in a strict sense. In fact, $\Pi_1 \subseteq \Pi_2$ only implies $\Pi_1^S \subseteq \Pi_2^S$ on instances of a certain minimum girth. To address this issue, we have to be careful about the girth in all three steps and can finally resolve the problem in the last step.

We now define the notion of girth. For an n-ary relation symbol S, $\mathsf{pos}(S)$ is $\{1, \dots, n\}$. A finite structure I has a *cycle* of length n if it contains distinct facts $R_0(\mathbf{a}_0), \dots, R_{n-1}(\mathbf{a}_{n-1})$, $\mathbf{a}_i = a_{i,1} \cdots a_{i,m_i}$, and there are positions $p_i, p'_i \in \mathsf{pos}(R_i)$, $0 \le i < n$ such that:

- $p_i \neq p'_i$ for $1 \le i \le n$;
- $a_{i,p'_i} = a_{i \oplus 1, p_{i \oplus 1}}$ for $0 \le i < n$, where $\oplus$ denotes addition modulo n.

The *girth* of I is the length of the shortest cycle in it and ∞ if I has no cycle (in which case we say that I is a *tree*).

For MDDLog programs Π_1, Π_2 over the same EDB schema and $k \ge 0$, we write $\Pi_1 \subseteq_{>k} \Pi_2$ if $\Pi_1(I) \subseteq \Pi_2(I)$ for all $\mathbf{S}$-instances of girth exceeding k.

Throughout the proof, we have to carefully analyze the running time of the algorithm, considering various measures for MDDLog programs. The *size* of an MDDLog program Π, denoted $|\Pi|$, is the number of symbols needed to write Π where relation and variable names are counted as having length one. The *rule size* of an MDDLog program is the maximum size of a rule in Π. The *atom width* (resp. *variable width*) of Π is the maximum number of atoms in any rule body (resp. variables in any rule) in Π.

From Unrestricted to Simple Programs An MDDLog program Π^S is *simple* if it satisfies the following conditions:

1. every rule in Π^S comprises at most one EDB atom and this atom contains all variables of the rule body, each variable exactly once;
2. rules without an EDB atom contain at most a single variable.

The conversion to simple form changes the EDB schema and thus the semantics of the involved queries, but it (almost) preserves containment, as detailed by the next theorem. The theorem is implicit in (Feder and Vardi 1998) and our contribution is to analyze the size of the constructed MDDLog programs and to provide detailed proofs. The same applies to the other theorems stated in this section.

Theorem 6. *Let Π_1, Π_2 be Boolean MDDLog programs over EDB schema $\mathbf{S}_E$. Then one can construct simple Boolean MDDLog programs Π_1^S, Π_2^S over EDB schema $\mathbf{S}'_E$ such that*

1. $\Pi_1 \not\subseteq \Pi_2$ implies $\Pi_1^S \not\subseteq \Pi_2^S$;

2. $\Pi_1^S \not\subseteq_{>w} \Pi_2^S$ implies $\Pi_1 \not\subseteq_{>w} \Pi_2$

where w is the atom width of $\Pi_1 \cup \Pi_2$. Moreover, if r is the number of rules in $\Pi_1 \cup \Pi_2$ and s the rule size, then

3. $|\Pi_i^S| \le p(r \cdot 2^s)$;

4. the variable width of Π_i^S is bounded by that of Π_i;

5. $|\mathbf{S}'_E| \le p(r \cdot 2^s)$;

where p is a polynomial. The construction takes time polynomial in $|\Pi_1^S \cup \Pi_2^S|$.

A detailed proof of Theorem 6 is given in the appendix. Here, we only sketch the construction, which consists of several steps. We concentrate on a single Boolean MDDLog program Π. In the first step, we extend Π with all rules that can be obtained from a rule in Π by consistently identifying variables. We then split up each rule in Π into multiple rules by introducing fresh IDB relations whenever this is possible. After this second step, we obtain a program which satisfies the following conditions:

(i) all rule bodies are biconnected, that is, when any single variable is removed from the body (by deleting all atoms that contain it), then the resulting rule body is still connected;

(ii) if $R(x, \dots, x)$ occurs in a rule body with R EDB, then the body contains no other EDB atoms.

In the final step, we transform every rule as follows: we replace all EDB atoms in the rule body by a single EDB atom that uses a fresh EDB relation which represents the conjunction of all atoms replaced. Additionally, we need to take care of implications between the new EDB relations, which gives rise to additional rules. The last step of the conversion is the most important one, and it is the reason for why we can only use instances of a certain girth in Point 2 of Theorem 6. Assume, for example, that, before the last step, the program had contained the following rules, where A and r are EDB relations:

$$P(x_3) \leftarrow A(x_1) \wedge r(x_1, x_2) \wedge r(x_2, x_3) \wedge r(x_3, x_1)$$
$$\mathsf{goal}() \leftarrow r(x_1, x_2) \wedge r(x_2, x_3) \wedge r(x_3, x_1) \wedge P(x_1) \wedge P(x_2) \wedge P(x_3)$$

A new ternary EDB relation R_{q_2} is introduced for the EDB body atoms of the lower rule, where $q_2 = r(x_1, x_2) \wedge r(x_2, x_3) \wedge r(x_3, x_1)$, and a new ternary EDB relation R_{q_1} is introduced for the upper rule, $q_1 = A(x_1) \wedge q_2$. Then the rules are replaced with

$$P(x_3) \leftarrow R_{q_1}(x_1, x_2, x_3)$$
$$\mathsf{goal}() \leftarrow R_{q_2}(x_1, x_2, x_3) \wedge P(x_1) \wedge P(x_2) \wedge P(x_3)$$
$$\mathsf{goal}() \leftarrow R_{q_1}(x_1, x_2, x_3) \wedge P(x_1) \wedge P(x_2) \wedge P(x_3)$$

Note that $q_1 \subseteq q_2$, which results in two copies of the goal rule to be generated. To understand the issues with girth, consider the $\mathbf{S}'_E$-instance I defined by

$$R_{q_1}(a, a', c'), R_{q_1}(b, b', a'), R_{q_1}(c, c', b').$$

The goal rules from the simplified program do not apply. But when translating into an $\mathbf{S}_E$-instance J in the obvious way, the goal rule of the original program does apply. The intuitive reason is that, when we translate J back to I, we get additional facts $R_{q_2}(a', b', c')$, $R_{q_2}(b', c', a')$, $R_{q_2}(c', a', b')$ that are 'missed' in I. Such effects can only happen on instances whose girth is at most w, such as I.

From Containment to Relativized Emptiness A *disjointness constraint* is a rule of the form $\bot \leftarrow P_1(\mathbf{x}) \wedge \cdots \wedge P_n(\mathbf{x})$ where all relations are of the same arity and at most unary. Let Π be a Boolean MDDLog program over EDB schema $\mathbf{S}_E$ and D a set of disjointness constraints over $\mathbf{S}_E$. We say that Π is *semi-simple w.r.t.* D if Π is simple when all relations that occur in D are viewed as IDB relations. We say that Π is *empty w.r.t.* D if for all $\mathbf{S}_E$-instances I with $I \models D$, we have $I \not\models \Pi$. The problem of relativized emptiness is to decide, given a Boolean MDDLog program Π and a set of disjointness constraints D such that Π is semi-simple w.r.t. D, whether Π is empty w.r.t. D.

Theorem 7. *Let Π_1, Π_2 be simple Boolean MDDLog programs over EDB schema $\mathbf{S}_E$. Then one can construct a Boolean MDDLog program Π over EDB schema $\mathbf{S}'_E$ and a set of disjointness constraints D over $\mathbf{S}'_E$ such that Π is semi-simple w.r.t. D and*

1. *if $\Pi_1 \not\subseteq \Pi_2$, then Π is non-empty w.r.t. D;*
2. *if Π is non-empty w.r.t. D on instances of girth $> g$, for some $g > 0$, then $\Pi_1 \not\subseteq_{>g} \Pi_2$;*

Moreover,

3. *$|\Pi| \leq |\Pi_1| \cdot 2^{|\mathbf{S}_{I,2}| \cdot v_1}$, $|D| \leq \mathcal{O}(|\mathbf{S}_{I,2}|)$;*
4. *the variable width of $\Pi \cup D$ is bounded by the variable width of $\Pi_1 \cup \Pi_2$;*
5. *$|\mathbf{S}'_E| \leq |\mathbf{S}_E| + |\Pi_2|$.*

where v_1 is the variable width of Π_1 and $\mathbf{S}_{I,2}$ is the IDB schema of Π_2. The construction takes time polynomial in $|\Pi \cup D|$.

Note that, in Point 2 of Theorem 7, girth one instances are excluded.

To prove Theorem 7, let Π_1, Π_2 be simple Boolean MDDLog programs over EDB schema $\mathbf{S}_E$. For $i \in \{1, 2\}$, let $\mathbf{S}_{I,i}$ be the set of IDB relations in Π_i with goal relations $\mathsf{goal}_i \in \mathbf{S}_{I,i}$ and assume w.l.o.g. that $\mathbf{S}_{I,1} \cap \mathbf{S}_{I,2} = \emptyset$. Set $\mathbf{S}'_E := \mathbf{S}_E \cup \mathbf{S}_{I,2} \cup \{\overline{P} \mid P \in \mathbf{S}_{I,2}\}$. The MDDLog program Π is constructed in two steps. We first add to Π every rule that can be obtained from a rule ρ in Π_1 by extending the rule body with

- $P(x)$ or $\overline{P}(x)$, for every variable x in ρ and every unary $P \in \mathbf{S}_{I,2}$, and
- $P()$ or $\overline{P}()$ for every nullary $P \in \mathbf{S}_{I,2}$; for $P = \mathsf{goal}_2$, we always include $\overline{P}()$ but never $P()$.

In the second step of the construction, we remove from Π every rule ρ whose body being true implies that a rule from Π_2 is violated, that is, there is a rule whose body is the CQ $q(\mathbf{x})$ and with head $P_1(\mathbf{y}_1) \vee \cdots \vee P_n(\mathbf{y}_n)$ and a variable substitution σ such that[2]

- $\sigma(q)$ is a subset of the body of ρ and
- $\overline{P_i}(\sigma(\mathbf{y}_i))$ is in the body of ρ, for $1 \leq i \leq n$.

The goal relation of Π is $\mathsf{goal}_1()$. The set of disjointness constraints D then consists of all rules $\bot \leftarrow P(x) \wedge \overline{P}(x)$ for each unary $P \in \mathbf{S}_{I,2}$ and $\bot \leftarrow P() \wedge \overline{P}()$ for each nullary $P \in \mathbf{S}_{I,2}$. It is not hard to verify that Π and D satisfy the size bounds from Theorem 7. We show in the appendix that Π satisfies Points 1 and 2 of Theorem 7.

[2]Of course, each $\mathbf{y}_i$ consists of either zero or one variable.

Deciding Relativized Emptiness We now show how to decide emptiness of an MDDLog program Π w.r.t. a set of disjointness constraints D assuming that Π is semi-simple w.r.t. D.

Theorem 8. *Given a Boolean MDDLog program Π over EDB schema $\mathbf{S}_E$ and a set of disjointness constraints D over $\mathbf{S}_E$ such that Π is semi-simple w.r.t. D, one can decide non-deterministically in time $\mathcal{O}(|\Pi|^3) \cdot 2^{\mathcal{O}(|D| \cdot v)}$ whether Π is empty w.r.t. D, where v is the variable width of Π.*

Let Π be a Boolean MDDLog program over EDB schema $\mathbf{S}_E$ and let D be a set of disjointness constraints over $\mathbf{S}_E$ such that Π is semi-simple w.r.t. D. To prove Theorem 8, we show how to construct a finite set of $\mathbf{S}_E$-instances satisfying D such that Π is empty w.r.t. D if and only if it is empty in the constructed set of instances. Let $\mathbf{S}_D$ be the set of all EDB relations that occur in D. For $i \in \{0, 1\}$, an *i-type* is a set t of i-ary relation symbols from $\mathbf{S}_D$ such that t does not contain all EDB relations that co-occur in a disjointness rule in D. The *0-type of an instance I* is the set θ of all nullary $P \in \mathbf{S}_D$ with $P() \in I$. For each constant a of I, we use t_a to denote the *1-type that a has in I*, that is, t_a contains all unary $P \in \mathbf{S}_D$ with $P(a) \in I$.

We build an $\mathbf{S}_E$-instance K_θ for each 0-type θ. The elements of K_θ are exactly the 1-types and K_θ consists of the following facts:

- $P(t)$ for each 1-type t and each $P \in t$;
- $R(t_1, \dots, t_n)$ for each relation $R \in \mathbf{S}_E \setminus \mathbf{S}_D$ and all 1-types $t_1, \dots, t_n$;
- $P()$ for each nullary $P \in \theta$.

Note that, by construction, K_θ is an $\mathbf{S}_E$-instance that satisfies all constraints in D.

Lemma 9. *Π is empty w.r.t. D iff $K_\theta \not\models \Pi$ for all 0-types θ.*

By Lemma 9, we can decide emptiness of Π by constructing all instances K_θ and then checking whether $K_\theta \not\models \Pi$. The latter is done by guessing an extension K'_θ of K_θ to the IDB relations in Π that does not contain the goal relation, and then verifying by an iteration over all possible homomorphisms from rule bodies in Π to K'_θ that all rules in Π are satisfied in K'_θ.

Lemma 10. *The algorithm for deciding relativized emptiness runs in time $\mathcal{O}(|\Pi|^3) \cdot 2^{\mathcal{O}(|D| \cdot v)}$.*

We still have to address the girth restrictions in Theorems 6 and 7, which are not reflected in Theorem 8. In fact, it suffices to observe that relativized emptiness is independent of the girth of witnessing structures. This is made precise by the following result.

Lemma 11. *For every Boolean MDDLog program Π over EDB schema $\mathbf{S}_E$ and set of disjointness constraints D over $\mathbf{S}_E$ such that Π is semi-simple w.r.t. D, the following are equivalent for any $g \geq 0$:*

1. *Π is empty regarding D and*

2. Π *is empty regarding D and instances of girth exceeding g.*

The proof of Lemma 11 uses a translation of semi-simple MDDLog programs with disjointness constraints into a constraint satisfaction problem (CSP) and invokes a combinatorial lemma by Feder and Vardi (and, originally, Erdős), to transform instances into instances of high girth while preserving certain homomorphisms.

Deriving Upper Bounds We exploit the results just obtained to derive upper complexity bounds, starting with Boolean MDDLog programs and MMSNP sentences. In the following theorem, note that for deciding $\Pi_1 \subseteq \Pi_2$, the contribution of Π_2 to the complexity is exponentially larger than that of Π_1.

Theorem 12. *Containment between Boolean MDDLog programs and between MMSNP sentences is in* 2NExpTime. *More precisely, for Boolean MDDLog programs Π_1 and Π_2, it can be decided non-deterministically in time $2^{2^{p(|\Pi_2|\cdot\log|\Pi_1|)}}$ whether $\Pi_1 \subseteq \Pi_2$, p a polynomial.*

We now extend Theorem 12 to MDDLog programs of unrestricted arity. Since this is easier to do when constants can be used in place of variables in rules, we actually generalize Theorem 12 by allowing both constants in rules and unrestricted arity. For clarity, we speak about MDDLog^c programs whenever we allow constants in rules. First, we show how to (Turing) reduce containment between MDDLog^c programs of unrestricted arity to containment between Boolean MDDLog^c programs. The idea essentially is to replace answer variables with fresh constants.

Let Π_1, Π_2 be MDDLog^c programs of arity k and let $\mathbf{C}$ be the set of constants in $\Pi_1 \cup \Pi_2$, extended with k fresh constants. We define Boolean MDDLog^c programs $\Pi_1^{\mathbf{a}}$, $\Pi_2^{\mathbf{a}}$ for each tuple $\mathbf{a}$ over $\mathbf{C}$ of arity k. If $\mathbf{a} = (a_1, \dots, a_k)$, then $\Pi_i^{\mathbf{a}}$ is obtained from Π_i by modifying each goal rule $\rho = \mathsf{goal}(\mathbf{x}) \leftarrow q$ with $\mathbf{x} = (x_1, \dots, x_k)$ as follows:

- if there are i, j such that $x_i = x_j$ and $a_i \neq a_j$, then discard ρ;
- otherwise, replace ρ with $\mathsf{goal}() \leftarrow q'$ where q' is obtained from q by replacing each x_i with a_i.

In the appendix, we show the following.

Lemma 13. $\Pi_1 \subseteq \Pi_2$ *iff* $\Pi_1^{\mathbf{a}} \subseteq \Pi_2^{\mathbf{a}}$ *for all* $\mathbf{a} \in \mathbf{C}^k$.

This provides the desired Turing reduction to the Boolean case, with constants. Note that the size of $\Pi_i^{\mathbf{a}}$ is bounded by that of Π_i, and likewise for all other relevant measures. The number of required containment tests is bounded by $2^{|\Pi_1 \cup \Pi_2|^2}$, a factor that is absorbed by the bounds in Theorem 12.

It remains to reduce containment between Boolean MDDLog^c programs to containment between Boolean MDDLog programs. The general idea is to replace constants with fresh monadic EDB relations. Of course, we have to be careful because the extension of these fresh relations in an instance need not be a singleton set. Let Π_1, Π_2 be Boolean MDDLog^c programs over EDB schema $\mathbf{S}_E$ and let $\mathbf{C}$ be the set of constants in $\Pi_1 \cup \Pi_2$. The EDB schema $\mathbf{S}'_E$ is obtained by extending $\mathbf{S}_E$ with a monadic relation R_a for each $a \in \mathbf{C}$. For $i \in \{1, 2\}$, the Boolean MDDLog program Π'_i over EDB schema $\mathbf{S}'_E$ contains all rules that can be obtained from a rule ρ from Π_i by choosing a partial function δ that maps the terms (variables and constants) in ρ to the relations in $\mathbf{S}'_E \setminus \mathbf{S}_E$ such that each constant a is mapped to R_a and then

1. replacing every occurrence of a term $t \in \mathsf{dom}(\delta)$ in the body of ρ with a fresh variable and every occurrence of t in the head of ρ with one of the variables introduced for t in the rule body;
2. adding $R_a(x)$ to the rule body whenever some occurrence of a variable x_0 in the original rule has been replaced with x and $\delta(x_0) = R_a$.

For example, the rule $P_1(y) \lor P_2(y) \leftarrow r(x, y, y) \land s(y, z)$ in Π_i gives rise, among others, to the following rule in Π'_i:

$$P_1(y_3) \lor P_2(y_1) \leftarrow r(x_1, y_1, y_2) \land s(y_3, z) \land R_{a_1}(x_1) \land R_{a_2}(y_1) \land R_{a_2}(y_2) \land R_{a_2}(y_3).$$

The above rule treats the case where the variable x from the original rule is mapped to the constant a_1, y to a_2, and z not to any constant in $\mathbf{C}$. Note that the original variables x and y have been duplicated because R_{a_1} and R_{a_2} need not be singletons while a_1 and a_2 denote a single object. So intuitively, Π'_i treats its input instance I as if it was the quotient I' of I obtained by identifying all a_1, a_2 with $R_b(a_1), R_b(a_2) \in I$ for some $b \in \mathbf{C}$. In addition to the above rules, Π'_2 also contains $\mathsf{goal}() \leftarrow R_{a_1}(x) \land R_{a_2}(x)$ for all distinct $a_1, a_2 \in \mathbf{C}$.

Lemma 14. $\Pi_1 \subseteq \Pi_2$ *iff* $\Pi'_1 \subseteq \Pi'_2$.

It can be verified that $|\Pi'_i| \leq 2^{|\Pi'_i|^2}$ and that the rule size of Π'_i is bounded by twice the rule size of Π_i. Because of the latter, the simplification of the programs Π'_i according to Theorem 6 yields programs whose size is still bounded by $2^{p(|\Pi_i|)}$, as in the proof of Theorem 12, and whose variable width is bounded by twice the variable width of Π_i. It is thus easy to check that we obtain the same overall bounds as stated in Theorem 12.

Theorem 15. *Containment between MDDLog programs of any arity and with constants is in* 2NExpTime. *More precisely, for programs Π_1 and Π_2, it can be decided non-deterministically in time $2^{2^{p(|\Pi_2|\cdot\log|\Pi_1|)}}$ whether $\Pi_1 \subseteq \Pi_2$, p a polynomial.*

Ontology-Mediated Queries

We now consider containment between ontology-mediated queries based on description logics, which we introduce next.

An *ontology-mediated query (OMQ)* over a schema $\mathbf{S}_E$ is a triple $(\mathcal{T}, \mathbf{S}_E, q)$, where $\mathcal{T}$ is a TBox formulated in a description logic and q is a query over the schema $\mathbf{S}_E \cup \mathsf{sig}(\mathcal{T})$, with $\mathsf{sig}(\mathcal{T})$ the set of relation symbols used in $\mathcal{T}$. The TBox can introduce symbols that are not in $\mathbf{S}_E$, which allows it to enrich the schema of the query q. As the TBox language, we use the description logic $\mathcal{ALC}$, its extension $\mathcal{ALCI}$ with inverse roles, and the further extension $\mathcal{SHI}$ of $\mathcal{ALCI}$ with transitive roles and role hierarchies. Since all these logics admit only unary and binary relations, we assume that these are the only allowed arities in schemas throughout the section. As the actual query language, we use UCQs and CQs. The OMQ languages that these choices give rise to are denoted with $(\mathcal{ALC}, \text{UCQ})$, $(\mathcal{ALCI}, \text{UCQ})$, $(\mathcal{SHI}, \text{UCQ})$, and so on. In OMQs $(\mathcal{T}, \mathbf{S}_E, q)$ from $(\mathcal{SHI}, \text{UCQ})$, we disallow superroles of transitive roles in q; it is known that allowing transitive roles in the query poses serious additional complications, which are outside the scope of this paper, see e.g. (Bienvenu et al. 2010; Gottlob, Pieris, and Tendera 2013). The semantics of an OMQ is given in terms of *certain answers*. We refer to the appendix for further details and only give an example of an OMQ from $(\mathcal{ALC}, \text{UCQ})$.

Example 16. *Let the OMQ $Q = (\mathcal{T}, \mathbf{S}_E, q)$ be given by*

$$\begin{aligned}\mathcal{T} = \{\ & \exists \mathsf{manages}.\mathsf{Project} \sqsubseteq \mathsf{Manager}, \\ & \mathsf{Employee} \sqsubseteq \mathsf{Male} \sqcup \mathsf{Female}\ \} \\ \mathbf{S}_E = \{ & \mathsf{Employee}, \mathsf{Project}, \mathsf{Male}, \mathsf{Female}, \mathsf{manages}\} \\ q(x) = {} & \mathsf{Manager}(x) \wedge \mathsf{Female}(x)\end{aligned}$$

On the $\mathbf{S}_E$-instance

$$\mathsf{manages}(e_1, e_2), \mathsf{Female}(e_1), \mathsf{Project}(e_2),$$
$$\mathsf{manages}(e_1', e_2), \mathsf{Employee}(e_1'),$$

the only certain answer to Q is e_1.

Let $Q_i = (\mathcal{T}_i, \mathbf{S}_E, q_i)$, $i \in \{1, 2\}$. Then Q_1 *is contained in* Q_2, written $Q_1 \subseteq Q_2$, if for every $\mathbf{S}_E$-instance I, the certain answers to Q_1 on I are a subset of the certain answers to Q_2 on I. The query containment problems between OMQs considered in (Bienvenu, Lutz, and Wolter 2012) are closely related to ours, but concern different (weaker) OMQ languages. One difference in setup is that, there, the definition of "contained in" does not refer to all $\mathbf{S}_E$-instances I, but only to those that are consistent with both $\mathcal{T}_1$ and $\mathcal{T}_2$. Our results apply to both notions of consistency. In fact, we show in the appendix that consistent containment between OMQs can be reduced in polynomial time to unrestricted containment as studied in this paper, and in our lower bound we use TBoxes that are consistent w.r.t. all instances. We use $|\mathcal{T}|$ and $|q|$ to denote the size of a TBox $\mathcal{T}$ and a query q, defined as for MDDLog programs.

The following is the main result on OMQs established in this paper. It solves an open problem from (Bienvenu, Lutz, and Wolter 2012).

Theorem 17. *The following containment problems are* 2NExpTime*-complete:*

1. *of an $(\mathcal{ALC}, UCQ)$-OMQ in a CQ;*
2. *of an $(\mathcal{ALC}, UCQ)$-OMQ in an $(\mathcal{ALC}, UCQ)$-OMQ;*
3. *of an $(\mathcal{ALCI}, CQ)$-OMQ in an $(\mathcal{ALCI}, CQ)$-OMQ;*
4. *of a $(\mathcal{SHI}, UCQ)$-OMQ in a $(\mathcal{SHI}, UCQ)$-OMQ.*

The lower bounds apply already when the TBoxes of the two OMQs are identical.

We start with the lower bounds. For Point 1 and 2 of Theorem 17, we make use of the lower bound that we have already obtained for MDDLog. It was observed in (Bienvenu et al. 2013; 2014) that both $(\mathcal{ALC}, \text{UCQ})$ and $(\mathcal{SHI}, \text{UCQ})$ have the same expressive power as MDDLog restricted to unary and binary EDB relations. In fact, every such MDDLog program can be translated into an equivalent OMQ from $(\mathcal{ALC}, \text{UCQ})$ in polynomial time. Thus, the lower bounds in Point 1 and 2 of Theorem 17 are a consequence of those in Theorem 3.

The lower bound stated in Point 3 of Theorem 17 is proved by a non-trivial reduction of the 2-exp torus tiling problem. Compared to the reduction that we have used for MDDLog, some major changes are required. In particular, the queries used there do not seem to be suitable for this case, and thus we replace them by a different set of queries originally introduced in (Lutz 2008). Details are in the appendix. It can be shown exactly as in the proof of Theorem 3 in (Bienvenu, Lutz, and Wolter 2012) that, in the lower bounds in Points 2 and 3, we can assume the TBoxes of the two OMQs to be identical.

We note in passing that we again obtain corresponding lower bounds for rewritability.

Theorem 18. *In $(\mathcal{ALC}, UCQ)$ and $(\mathcal{ALCI}, CQ)$, rewritability into FO, into monadic Datalog, and into Datalog is* 2NExpTime*-hard.*

Now for the upper bounds in Theorem 17. The translation of OMQs into MDDLog programs is more involved than the converse direction, a naive attempt resulting in an MDDLog program with existential quantifiers in the rule heads. We next analyze the blowups involved. The construction used in the proof of the following theorem is a refinement of a construction from (Bienvenu et al. 2013), resulting in improved bounds.

Theorem 19. *For every OMQ $Q = (\mathcal{T}, \mathbf{S}_E, q)$ from $(\mathcal{SHI}, UCQ)$, one can construct an equivalent MDDLog program Π such that*

1. *$|\Pi| \leq 2^{2^{p(|q| \cdot \log |\mathcal{T}|)}}$;*
2. *the IDB schema of Π is of size $2^{p(|q| \cdot \log |\mathcal{T}|)}$;*
3. *the rule size of Π is bounded by $|q|$*

where p is a polynomial. The construction takes time polynomial in $|\Pi|$.

We now use Theorem 19 to derive an upper complexity bound for containment in $(\mathcal{SHI}, \text{UCQ})$. While there are double exponential blowups both in Theorem 15 and in Theorem 19, a careful analysis reveals that they do not add up and, overall, still give rise to a 2NExpTime upper bound. In contrast to Theorem 12, though, we only get an algorithm whose running time is double exponential in both inputs $(\mathcal{T}_1, \mathbf{S}_E, q_1)$ and $(\mathcal{T}_2, \mathbf{S}_E, q_2)$. However, it is double exponential only in the size of the actual queries q_1 and q_2 while being only single exponential in the size of the TBoxes $\mathcal{T}_1$ and $\mathcal{T}_2$. This is good news since the size of q_1 and q_2 is typically very small compared to the sizes of $\mathcal{T}_1$ and $\mathcal{T}_2$. For this reason, it can even be reasonable to assume that the sizes of q_1 and q_2 are constant, in the same way in which the size of the query is assumed to be constant in classical data complexity. Note that, under this assumption, we obtain a NExpTime upper bound for containment.

Theorem 20. *Containment between OMQs from $(\mathcal{SHI}, UCQ)$ is in* 2NExpTime. *More precisely, for OMQs $Q_1 = (\mathcal{T}_1, \mathbf{S}_E, q_1)$ and $Q_2 = (\mathcal{T}_2, \mathbf{S}_E, q_2)$, it can be decided non-deterministically in time $2^{2^{p(|q_1|\cdot|q_2|\cdot\log|\mathcal{T}_1|\cdot\log|\mathcal{T}_2|)}}$ whether $Q_1 \subseteq Q_2$, p a polynomial.*

Outlook

There are several interesting questions left open. One is whether decidability of containment in MMSNP generalizes to GMSNP, where IDB relations can have any arity and rules must be frontier-guarded (Bienvenu et al. 2014) or even to frontier-guarded disjunctive TGDs, which are the extension of GMSNP with existential quantification in the rule head (Bourhis, Morak, and Pieris 2013). We remark that an extension of Theorem 19 to frontier-*one* disjunctive TGDs (where rule body and head share only a single variable) seems not too hard.

Other open problems concern containment between OMQs. In particular, it would be good to know the complexity of containment in $(\mathcal{ALC}, \text{CQ})$ which must lie between NExpTime and 2NExpTime. Note that our first lower bound crucially relies on *unions* of conjunctive queries to be evailable, and the second one on inverse roles. It is known that adding inverse roles to $\mathcal{ALC}$ tends to increase the complexity of querying-related problems (Lutz 2008), so the complexity of containment in $(\mathcal{ALC}, \text{CQ})$ might indeed be lower than 2NExpTime. It would also be interesting to study containment for OMQs from $(\mathcal{ALCI}, \text{UCQ})$ where the actual query is connected and has at least one answer variable. In the case of query answering, such a (practically very relevant) assumption causes the complexity to drop (Lutz 2008). Is this also the case for containment?

Acknowledgements. Bourhis was supported by CPER Data and Lutz was supported by ERC grant 647289 CODA.

References

Afrati, F. N.; Cosmadakis S. S., Yannakakis, M. 1995. On Datalog vs. Polynomial Time. *J. Comput. Syst. Sci.* 51(2):177–196.

Benedikt, M.; Bourhis, P.; and Senellart, P. 2012. Monadic datalog containment. In *Proc. of ICALP2012*, volume 7392 of *LNCS*, 79–91. Springer.

Bienvenu, M.; Eiter, T.; Lutz, C.; Ortiz, M.; and Simkus, M. 2010. Query answering in the description logic $\mathcal{S}$. In *Proc. of DL2010*, volume 573 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Bienvenu, M.; ten Cate, B.; Lutz, C.; and Wolter, F. 2013. Ontology-based data access: a study through disjunctive Datalog, CSP, and MMSNP. In *Proc. of PODS2013*, 213–224. ACM.

Bienvenu, M.; ten Cate, B.; Lutz, C.; and Wolter, F. 2014. Ontology-based data access: A study through disjunctive Datalog, CSP, and MMSNP. *ACM Trans. Database Syst.* 39(4):33:1–33:44.

Bienvenu, M.; Lutz, C.; and Wolter, F. 2012. Query containment in description logics reconsidered. In *Proc. of KR2012*. AAAI Press.

Björklund, H.; Martens, W.; and Schwentick, T. 2008. Optimizing conjunctive queries over trees using schema information. In *Proc. of MFCS2008*, volume 5162 of *LNCS*, 132–143. Springer.

Bodirsky, M., and Dalmau, V. 2013. Datalog and constraint satisfaction with infinite templates. *J. Comput. Syst. Sci.* 79(1):79–100.

Bodirsky, M.; Chen, H.; and Feder, T. 2012. On the complexity of MMSNP. *SIAM J. Discrete Math.* 26(1):404–414.

Bourhis, P.; Krötzsch, M.; and Rudolph, S. 2015. Reasonable highly expressive query languages. In *Proc. of IJCAI2015*, 2826–2832. AAAI Press.

Bourhis, P.; Morak, M.; and Pieris, A. 2013. The impact of disjunction on query answering under guarded-based existential rules. In *Proc. of IJCAI2013*. IJCAI/AAAI.

Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; Poggi, A.; Rodriguez-Muro, M.; and Rosati, R. 2009. Ontologies and databases: The DL-Lite approach. In *Proc. of Reasoning Web 2009*, volume 5689 of *LNCS*, 255–356. Springer.

Calvanese, D.; De Giacomo, G.; and Lenzerini, M. 1998. On the decidability of query containment under constraints. In *Proc. of PODS1998*, 149–158. ACM Press.

Calvanese, D.; Ortiz, M.; and Simkus, M. 2011. Containment of regular path queries under description logic constraints. In *Proc. of IJCAI2011*, 805–812. IJCAI/AAAI.

Cosmadakis, S. S.; Gaifman, H.; Kanellakis, P. C.; and Vardi, M. Y. 1988. Decidable optimization problems for database logic programs (preliminary report). In *Proc. of STOC1988*, 477–490. ACM.

Eiter, T.; Gottlob, G.; and Mannila, H. 1997. Disjunctive Datalog. *ACM Trans. Database Syst.* 22(3):364–418.

Feder, T., and Vardi, M. Y. 1998. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.* 28(1):57–104.

Gottlob, G.; Pieris, A.; and Tendera, L. 2013. Querying the guarded fragment with transitivity. In *Proc. of ICALP2013*, volume 7966 of *LNCS*, 287–298. Springer.

Larose, B.; Loten, C.; and Tardif, C. 2007. A characterisation of first-order constraint satisfaction problems. *Logical Methods in Computer Science* 3(4).

Levy, A. Y., and Rousset, M. 1996. CARIN: A representation language combining Horn rules and description logics. In *Proc. of ECAI1996*, 323–327. John Wiley and Sons, Chichester.

Lutz, C. 2008. The complexity of conjunctive query answering in expressive description logics. In *Proc. of IJCAR2008*, volume 5195 of *LNCS*, 179–193. Springer.

Madelaine, F. R. 2010. On the containment of forbidden patterns problems. In *Proc. of CP2010*, volume 6308 of *LNCS*, 345–359. Springer.

Shmueli, O. 1993. Equivalence of Datalog queries is undecidable. *J. Log. Program.* 15(3):231–241.

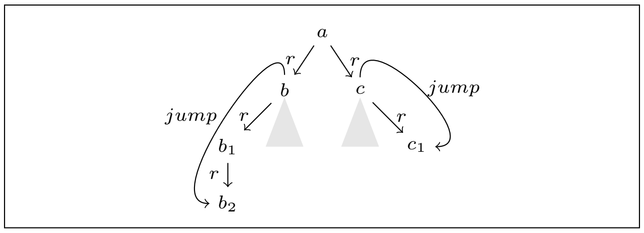


Figure 3: The counting gadgets.

MDDLog Hardness: Missing Details

We first repeat the details of the construction of a UCQ which achieves (Q1) along with additional information, then prove Lemma 4, and subsequently describe how the UCQ can be replaced by a CQ. Finally, we prove Theorem 5. Since the constructed queries will make use of the counting gadgets in counting trees, we show these gadgets again in Figure 3. There, a is a node in a counting tree, b is its left successor, and c is its right successor.

To define the UCQ that achieves (Q1), set $q_0(x, y) = r(x_0, y_0)$ and, for $0 < i \leq m$,

$$\begin{aligned} q_i(x_i,y_i) = {} & \exists x_{i-1} \exists y_{i-1} \exists z_{i,0} \cdots \exists z_{i,i+2} \exists z'_{i,1} \cdots \exists z'_{i,i+3} \\ & q_{i-1}(x_{i-1}, y_{i-1}) \wedge r(x_{i-1}, x_i) \wedge r(y_{i-1}, y_i) \wedge \\ & \mathsf{jump}(x_i, z_{i,i+2}) \wedge \mathsf{jump}(y_i, z'_{i,i+3}) \wedge \\ & r(z_{i,0}, z_{i,1}) \wedge \cdots \wedge r(z_{i,i+1}, z_{i,i+2}) \wedge \\ & r(z_{i,0}, z'_{i,1}) \wedge r(z_{i,1}, z'_{i,2}) \wedge \cdots \wedge r(z'_{i,i+2}, z'_{i,i+3}) \end{aligned}$$

The idea is that $I \models q_m[a, b]$ if a and b are leafs in successive counting trees that are at the same leaf position, that is, (i) the roots of the trees are connected by the relation r and (ii) a can be reached from the root of the first tree by following the same sequence of left and right successors that one also needs to follow to reach b from the root of the second tree. In fact, the r-atom in q_0 corresponds to the move from the root of one counting tree to the root of a successive tree, the atoms $r(x_{i-1}, x_i)$ and $r(y_{i-1}, y_i)$ in q_i correspond to moving down the i-th step in both trees, and the remaining atoms in q_i make sure that both of these steps are to a left successor or to a right successor.

To understand the latter, note that jump is the relation used in the navigation gadgets attached to tree nodes. The variable $z_{i,i+2}$ can only be mapped to the target of the jump relation in the navigation gadget at x_i, and likewise for $z'_{i,i+3}$ and the target of the jump relation in the navigation gadget at y_i. Note that there must be a $z_{i,0}$ from which $z_{i,i+2}$ can be reached along an r-path of length $i + 2$ and from which $z'_{i,i+3}$ can be reached along an r-path of length $i + 3$. If x_i and y_i are both left successors, then this $z_{i,0}$ is the root of the first counting tree. If x_i and y_i are both right successors, then $z_{i,0}$ is the r-predecessor of the root of the first counting tree, which must exist at all relevant nodes: at grid nodes because of the self step nodes and at horizontal/vertical step nodes because they have a grid node as r-predecessor. If x_i is a left successor and y_i a right successor or vice versa, then there is no target for $z_{i,0}$ because this target would have to reach the root of the first counting tree on a path of length one and on a path of length zero, but there is no reflexive loop at the root of counting trees (only a length two loop via self step nodes).

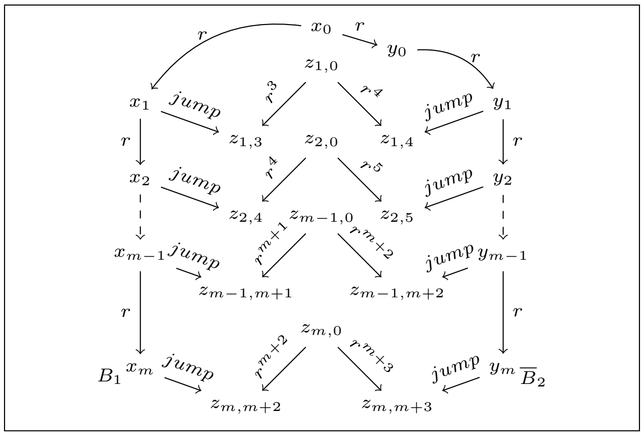


Figure 4: The first CQ in q (again).

It is now easy to define the desired UCQ:

$$\begin{aligned} q = {} & \exists x_m \exists y_m \, q_m(x_m, y_m) \wedge B_1(x_m) \wedge \overline{B}_2(y_m) \\ & \vee \exists x_m \exists y_m \, q_m(x_m, y_m) \wedge \overline{B}_1(x_m) \wedge B_2(y_m) \end{aligned}$$

The first CQ in this UCQ is displayed in Figure 4. As required, it evaluates to true on an instance if there are successive counting trees (whose roots have an r-predecessor and) which contain two leafs at the same position that are labeled differently regarding $B_1, \overline{B}_1$ and $B_2, \overline{B}_2$. This finishes the construction for the case of UCQs. We first establish correctness and then show how to replace the UCQ with a CQ.

Lemma 4. $\Pi \not\subseteq q$ *iff there is no tiling for* P *and* w_0.

Proof.(sketch) Assume first that there is no tiling for P and w_0. Let I be the instance that represents the $2^{2^n} \times 2^{2^n}$-grid with counting trees in the way described above. It can be verified that $I \not\models q$. We aim to show that $I \models \Pi$ and thus I witnesses $\Pi \not\subseteq q$. Assume to the contrary that $I \not\models \Pi$. Then there is an extension J of I that satisfies all rules in Π, but does not contain $\mathsf{goal}()$. In particular, J must contain at least one atom $T_i(c)$ for each constant c with $\mathsf{gactive}(c) \in J$, thus we can choose a concrete $T_i(c)$ for each such c. Since none of the goal rules in Π applies, these chosen atoms must represent a tiling for P and w_0. We have thus obtained a contradiction to the assumption that no such tiling exists.

Now assume that there is a tiling f for P and w_0. Take an instance I with $I \models \Pi$. Assume to the contrary of what is to be shown that $I \not\models q$. Then I satisfies Conditions (Q1) and (Q2). Extend I to a new instance J as follows. Since I satisfies (Q2), every g-active constant c in I is associated with a unique counter value, thus with a unique horizontal position $x \in \{0, \ldots, 2^{2^n} - 1\}$ and a unique vertical position $y \in \{0, \ldots, 2^{2^n} - 1\}$. Include $T_i(c) \in J$ if $f(x, y) = T_i$ and then exhaustively apply all non-disjunctive rules from Π_{tree}. One can verify that J satisfies all rules in Π while making the goal relation false, in contradiction to $I \models \Pi$. In particular, satisfaction of (Q1) and the way in which we have added facts $T_i(c)$ to J imply that none of the goal rules that check for a tiling defect applies. ❑

To replace the UCQ q by a CQ, we again use a coding trick from (Björklund, Martens, and Schwentick 2008). The basic idea is to replace $B_1, \overline{B}_1$ and $B_2, \overline{B}_2$ with suitable *bit gadgets* and then to use a construction that is very similar to the one used above for ensuring that we consistenly follow left successors or right successors in corresponding steps of the navigation in the two involved trees.

We replace $B_1(x)$ with the following *bit one gadget*:

$$r(x, x_1) \wedge r(x_1, x_2) \wedge r(x_2, x_3) \wedge r(x_3, x_4) \wedge \mathsf{jump}_1(x, x_1) \wedge \mathsf{jump}_1(x, x_4)$$

where jump_1 is a fresh EDB relation and $\overline{B}_1(x)$ with the following *bit zero gadget*:

$$r(x, x_1) \wedge r(x_1, x_2) \wedge r(x_2, x_3) \wedge r(x_3, x_4) \wedge \mathsf{jump}_1(x, x_2) \wedge \mathsf{jump}_1(x, x_3).$$

B_2 and $\overline{B}_2$ are replaced with corresponding gadgets in which only jump_1 is replaced with jump_2. The existence of these bit gadgets needs to be verified in the rules of Π_{tree} that ensure the existence of counting trees. In addition to that, we require one further modification to Π_{tree}: the self step loops of length two at each grid node are replaced with self step nodes of length four. All three intermediate nodes on these loops behave exactly like a self step node before. We then replace the above UCQ by

$$\begin{aligned} q = {} & \exists x_m \exists y_m \exists z_0 \cdots \exists z_{m+2} \exists z'_1 \cdots \exists z_{m+5} \\ & q_m(x_m, y_m) \wedge \\ & \mathsf{jump}_1(x_m, z_{m+2}) \wedge \mathsf{jump}_2(y_m, z'_{m+5}) \wedge \\ & r(z_0, z_1) \wedge \cdots \wedge r(z_{m+1}, z_{m+2}) \wedge \\ & r(z_0, z'_1) \wedge r(z_1, z'_2) \wedge \cdots \wedge r(z'_{m+4}, z'_{m+5}) \end{aligned}$$

Note that the x_m and y_m must be leafs in successive counting trees with the same leaf position. Additionally, the variable z_{m+2} can only be mapped to a target of the jump_1 relation in the bit gadget at x_m, and likewise for z'_{m+5} and a target of the jump_2 relation in the bit gadget at y_m. There must also be a z_0 from which z_{m+2} can be reached along an r-path of length $m + 2$ and from which z'_{m+5} can be reached along an r-path of length $m + 5$ (thus the difference in lengths is three). If the bit value at x_m is zero and the bit value at y_m is one, then this z_0 is the root of the first counting tree. If the bit value at x_m is one and the bit value at y_m is zero, the we can use for z_0 an r-predecessor of the root of the first counting tree. It can be verified that when the bit values at x_m and y_m are identical, then the possible target for z_0 must have r-paths to the root of the first counting tree of length i and j steps, for some

$$(i, j) \in \{(0, 2), (0, 3), (1, 0), (1, 3)\}.$$

However, since we have extended the length of self loops at grid nodes from two to four, there is no such target. This finishes the construction of the CQ q and establishes the lower bounds stated in Theorem 3.

We now come to the proof of Theorem 5.

Theorem 5. *For MDDLog programs and the complements of MMSNP sentences, rewritability into FO, into monadic Datalog, and into Datalog are* 2NExpTime*-hard.*

Proof. It suffices to consider Boolean MDDLog programs. First note that, by Rossman's theorem, any such program that is rewritable into FO is rewritable into a UCQ. Consequently, FO-rewritability implies monadic Datalog-rewritability implies Datalog-rewritability. Based on this observation, we deal with all three kinds of rewritability in a single proof: we show that from a 2-exp square tiling problem P and an input w_0 to P, we can construct in polynomial time a Boolean MDDLog program Π' such that

1. if there is a tiling for P and w_0, then Π' is FO-rewritable;
2. if there is no tiling for P and w_0, then Π' is not Datalog-rewritable.

Reconsider the reduction of the 2-exp square tiling problem to MDDLog containment given above. Given a 2-exp square tiling problem P and an input w_0 to P, we have shown how to construct a Boolean MDDLog program Π and a Boolean CQ q such that $\Pi \subseteq q$ iff there is a tiling for P and w_0. Let $\mathbf{S}_E$ be the EDB schema of Π and q. To obtain the desired program Π', we modify Π as follows:

1. in every goal rule, change the head to $A(x)$;
2. add $\mathsf{goal}() \leftarrow q$,
3. add $R(x) \vee G(x) \vee B(x) \leftarrow A(x)$, $\mathsf{goal}() \leftarrow C_1(x) \wedge C_2(x)$ for all distinct $C_1, C_2 \in \{R, G, B\}$, and $\mathsf{goal}() \leftarrow C(x) \wedge s(x, y) \wedge C(y)$ for all $C \in \{R, G, B\}$

where s is a fresh EDB relation and A, R, G, B are IDB relations. Let $\mathbf{S}'_E = \mathbf{S}_E \cup \{s\}$. We now show that Π' satisfies Points 1 and 2 above.

For Point 1, assume that there is a tiling for P and w_0. Then $\Pi \subseteq q$. We claim that q is a rewriting of Π'. By construction of Π', $I \models q$ clearly implies $I \models \Pi'$ for all $\mathbf{S}'_E$-instances I. For the converse, let $I \models \Pi'$. First assume $I \not\models \Pi$. Then there is an extension J of I to the IDB relations in Π' such that the extension of A is empty. Consequently, we must have $I \models \Pi'$ because $I \models q$ and we are done. Now assume $I \models \Pi$. Since $\Pi \subseteq q$, this implies $I \models q$ as desired.

For Point 2, assume there is no tiling for P and w_0. Then $\Pi \not\subseteq q$. Given an undirected graph $G = (V, E)$, let the instance I_G^+ be defined as the disjoint union of the instance I_0 which represents the 2^{2^n}-grid plus counting gadgets and the instance I_G that contains the fact $s(v_1, v_2)$ for every $\{v_1, v_2\} \in V$.

Since there is no tiling for P and w_0, we have $I_0 \models \Pi$ and thus $I_G^+ \models \Pi$. By construction of Π' and since $\Pi \not\models q$, this implies that $I_G \models \Pi'$ iff G is not 3-colorable. Assume to the contrary of what is to be shown that there is a Datalog-rewriting Γ of Π'. It is not difficult to modify Γ so that its EDB schema is $\mathbf{S}''_E = \{s\}$ and on any $\mathbf{S}''_E$-instance I_G representing an undirected graph G, the modified program Γ' yields the same result that Γ yields on I_G^+. We only sketch the idea: for every n-ary IDB relation S of Γ, all sets of positions $P = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$, and all tuples $t = (d_1, \ldots, d_k)$ of elements of I_0, introduce a fresh $n - k$-ary IDB $S_{P,t}$. Intuitively, $S_{P,t}$ is used to represent facts $S(d_1, \ldots, d_k)$ where every position $i_j \in P$ is mapped to the element d_j (which is not part of the input instance) and every position that is not in P is mapped to an element of the input instance I_G. It remains to introduce additional versions of each rule that use the new predicates, in all possible combinations.

We have thus shown that non-3-colorability of graphs can be expressed in Datalog, which is not the case (Afrati et al. 1995). ❑

MDDLog Upper Bound: Missing Details

From Unrestricted to Simple Programs

Theorem 6. *Let Π_1, Π_2 be Boolean MDDLog programs over EDB schema $\mathbf{S}_E$. Then one can construct simple Boolean MDDLog programs Π_1^S, Π_2^S over EDB schema $\mathbf{S}'_E$ such that*

1. *$\Pi_1 \not\subseteq \Pi_2$ implies $\Pi_1^S \not\subseteq \Pi_2^S$;*
2. *$\Pi_1^S \not\subseteq_{>w} \Pi_2^S$ implies $\Pi_1 \not\subseteq_{>w} \Pi_2$*

where w is the atom width of $\Pi_1 \cup \Pi_2$. Moreover, if r is the number of rules in $\Pi_1 \cup \Pi_2$ and s the rule size, then

3. *$|\Pi_i^S| \leq p(r \cdot 2^s)$;*
4. *the variable width of Π_i^S is bounded by that of Π_i;*
5. *$|\mathbf{S}'_E| \leq p(r \cdot 2^s)$;*

where p is a polynomial. The construction takes time polynomial in $|\Pi_1^S \cup \Pi_2^S|$.

To prove Theorem 6, we first concentrate on a single Boolean MDDLog program Π over EDB schema $\mathbf{S}_E$. We first construct from Π an equivalent MDDLog program Π' such that the following conditions are satisfied:

(i) all rule bodies are biconnected, that is, when any single variable is removed from the body (by deleting all atoms that contain it), then the resulting rule body is still connected;

(ii) if $R(x, \ldots, x)$ occurs in a rule body with R EDB, then the body contains no other EDB atoms.

To construct Π', we first extend Π with all rules that can be obtained from a rule in Π by consistently identifying variables and then exhaustively apply the following rules:

- replace every rule $p(\mathbf{y}) \leftarrow q_1(\mathbf{x}_1) \wedge q_2(\mathbf{x}_2)$ where $\mathbf{x}_1$ and $\mathbf{x}_2$ share exactly one variable x but both contain also other variables with the rules $p_1(\mathbf{y}_1) \vee Q(x) \leftarrow q_1(\mathbf{x}_1)$ and $p_2(\mathbf{y}_2) \leftarrow Q(x) \wedge q_2(\mathbf{x}_2)$, where Q is a fresh monadic IDB relation and $p_i(\mathbf{y}_i)$ is the restriction of $p(\mathbf{y})$ to atoms that are nullary or contain a variable from q_i, $i \in \{1, 2\}$;
- replace every rule $p(\mathbf{y}) \leftarrow q_1(\mathbf{x}_1) \wedge q_2(\mathbf{x}_2)$ where $\mathbf{x}_1$ and $\mathbf{x}_2$ share no variables and are both non-empty with the rules $p_1(\mathbf{y}_1) \vee Q() \leftarrow q_1(\mathbf{x}_1)$ and $p_2(\mathbf{y}_2) \leftarrow Q() \wedge q_2(\mathbf{x}_2)$, where $Q()$ is a fresh nullary IDB relation and the $p_i(\mathbf{y}_i)$ are as above;
- replace every rule $p(\mathbf{y}) \leftarrow R(x, \ldots, x) \wedge q(\mathbf{x})$ where R is an EDB relation and q contains at least one EDB atom and the variable x, with the rules $Q(x) \leftarrow R(x, \ldots, x)$ and $p(\mathbf{y}) \leftarrow Q(x) \wedge q(\mathbf{x})$, where Q is a fresh monadic IDB relation.

It is easy to see that the program Π' is equivalent to the original program Π. We next construct from Π' the desired simplification Π^S of Π. The intuition is that in every rule of Π', we replace all EDB atoms in the rule body by a single EDB atom that uses a fresh EDB relation which represents the conjunction of all atoms replaced. We also need to take care of implications between the new EDB relations. In the following, we make the construction precise.

For every conjunctive query $q(\mathbf{x})$ and schema $\mathbf{S}$, we use $q(\mathbf{x})|_{\mathbf{S}}$ to denote the restriction of $q(\mathbf{x})$ to $\mathbf{S}$-atoms. The EDB schema $\mathbf{S}'_E$ of Π^S consists of the relations $R_{q(\mathbf{x})|_{\mathbf{S}_E}}$, $p(\mathbf{y}) \leftarrow q(\mathbf{x})$ a rule in Π'; the arity of $R_{q(\mathbf{x})|_{\mathbf{S}_E}}$ is the number of variables in $q(\mathbf{x})$ (equivalently: in $q(\mathbf{x})|_{\mathbf{S}_E}$).

Let $\mathbf{S}_I$ be the IDB schema of Π'. The program Π^S consists of the following rules:

($*$) whenever $p(\mathbf{y}) \leftarrow q_1(\mathbf{x}_1)$ is a rule in Π', $R_{q_2(\mathbf{x}_2)}$ an EDB relation in $\mathbf{S}'_E$, and $h : \mathbf{x}_1 \rightarrow \mathbf{x}_2$ an injective homomorphism from $q_1(\mathbf{x}_1)$ to $q_2(\mathbf{x}_2)$, then Π^S contains the rule $p(\mathbf{y}) \leftarrow h^{-1}(R_{q_2}(\mathbf{x}_2)) \wedge q_1(\mathbf{x}_1)|_{\mathbf{S}_I}$

where $h^{-1}(R_{q_2}(\mathbf{x}_2))$ denotes the result of replacing in $R_{q_2}(\mathbf{x}_2)$ every variable x with y if $x = h(y)$ and every variable that does not occur in the range of h with a fresh variable. The case where $q_1(\mathbf{x}_1)$ is identical to $q_2(\mathbf{x}_2)$ and h is the identity corresponds to adapting rules in Π' to the new EDB signature and the other cases take care of implications between EDB relations, as announced.

The last step of the conversion just described is the most important one, and it is the reason for why we can only use instances of a certain girth in Point 2 of Theorem 6. An example illustarting this step and the issue with girth can be found in the main part of the paper. We next analyze the size of the constructed program Π^S.

Lemma 21. *Let r be the number of rules in Π and s the rule size of Π. Then there is a polynomial p such that*

1. *$|\Pi^S| \leq p(r \cdot 2^s)$;*
2. *the variable width of Π^S is bounded by that of Π;*
3. *$|\mathbf{S}'_E| \leq p(r \cdot 2^s)$.*

Proof. We start with Point 1. Let r be the number of rules in Π, v the variable width, and w the atom width. Regarding the size of Π^S, note that the first (identification) step replaces each rule of Π with at most $k!$ rules, where k is the number of variables in the original rule. After this step, we thus have at most $r \cdot v!$ rules. The subsequent rewriting in the construction of Π' splits each rule into at most one rule per atom in the original rule. The number of rules in Π' is thus bounded by $r \cdot v! \cdot w$. The number of rules in Π^S is clearly at most quadratic in the number of rules in Π', thus their number is bounded by $(r \cdot v! \cdot w)^2$. None of the steps increases the rule size, i.e., the rule size of Π^S is bounded by the rule size of Π. This yields the bound stated in Point 1.

Point 2 is very easy to verify by analyzing the construction of Π^S.

For Point 3, note that the program Π' has the same EDB schema as Π. In the construction of Π^S, the number of EDB relations is bounded by the number of rules in Π', thus by $r \cdot v! \cdot w$. ❑

So far, we have concentrated on a single program. To obtain Theorem 6, we have to jointly simplify that two involved programs Π_1 and Π_2. This only means that, when constructing Π_i^S from Π'_i in the second step of the normalization procedure, then we use the set of EDB relations introduced for *both* Π'_1 and Π'_2 instead of only those for Π'_i. The bounds in Lemma 21 then cleary give rise to those in Theorem 6. It remains to show that the joint simplification Π_1^S, Π_2^S of Π_1, Π_2 behaves as expected regarding containment.

Lemma 22.
1. $\Pi_1 \not\subseteq \Pi_2$ *implies* $\Pi_1^S \not\subseteq \Pi_2^S$*;*
2. $\Pi_1^S \not\subseteq_{>w} \Pi_2^S$ *implies* $\Pi_1 \not\subseteq \Pi_2$.

where w is the atom width of $\Pi_1 \cup \Pi_2$.

Proof. It is obvious that the construction of the program Π'_i from Π_i preserves equivalence. We can therefore assume that the programs Π_1 and Π_2 in Lemma 22 are in fact the programs Π'_1 and Π'_2. For $i \in \{1, 2\}$, let $\mathbf{S}_{I,i}$ be the IDB-schema of Π_i (and thus also of Π_i^S), and let goal_i be the goal relation of Π_i.

For Point 1, let I be an $\mathbf{S}_E$-instance such that $I \models \Pi_1$ and $I \not\models \Pi_2$. Let J be the $\mathbf{S}'_E$-instance that consists of all facts $R_q(a_1, \dots, a_n)$ such that $I \models q[a_1, \dots, a_n]$. It remains to show that $J \models \Pi_1^S$ and $J \not\models \Pi_2^S$.

For $J \models \Pi_1^S$, assume to the contrary of what is to be shown that there is an extension J' of J to schema $\mathbf{S}'_E \cup \mathbf{S}_{I,1}$ that satisfies all rules of Π_i^S and does not contain the goal_1 relation. Let I' be the corresponding extension of I, that is, I' extends I with the $\mathbf{S}_{I,1}$-facts from J'. It suffices to show that J' satisfies all rules in Π_1 to obtain a contradiction against $I \models \Pi_1$. Thus, let $p(\mathbf{y}) \leftarrow q(\mathbf{x})$ be a rule in Π_1 and let h be a homomorphism from $q(\mathbf{x})$ to I'. Then Π_1^S contains the rule $p(\mathbf{y}) \leftarrow R_{q(\mathbf{x})|_{\mathbf{S}_E}}(\mathbf{x}) \wedge q(\mathbf{x})|_{\mathbf{S}_{I_1}}$. By construction of J' and I', h is also a homomorphism from $R_{q(\mathbf{x})|_{\mathbf{S}_E}}(\mathbf{x}) \wedge q(\mathbf{x})|_{\mathbf{S}_{I_1}}$ to J'. Since J' satisfies all rules of Π_1^S, one of the disjuncts of $p(\mathbf{y})$ is satisfied under h, as required.

For $J \not\models \Pi_2^S$, let I' be an extension of I to $\mathbf{S}_E \cup \mathbf{S}_{I,2}$ that satisfies all rules of Π_2 and does not contain the goal_2 relation. Let J' be the corresponding extension of J. It suffices to show that J' satisfies all rules of Π_2^S. Thus, let $p(\mathbf{y}) \leftarrow q(\mathbf{x})$ be a rule in Π_2^S and let h be a homomorphism from $q(\mathbf{x})$ to J'. Then there is a rule $p(\mathbf{y}) \leftarrow q'(\mathbf{x}')$ in Π_2, a relation $R_{q''(\mathbf{x}'')}$ in $\mathbf{S}'_E$, and an injective homomorphism g from $q'(\mathbf{x}')$ to $q''(\mathbf{x}'')$ such that $q(\mathbf{x}) = g^{-1}(R_{q''(\mathbf{x}'')}(\mathbf{x}'')) \wedge q'(\mathbf{x}')|_{\mathbf{S}_{I_2}}$. By construction of J and J', h is also a homomorphism from $q'(\mathbf{x}')$ to I. Since I' satisfies $p(\mathbf{y}) \leftarrow q'(\mathbf{x}')$, one of the disjuncts of $p(\mathbf{y})$ is satisfied under h, as required.

Now for Point 2. Let I be an $\mathbf{S}'_E$-instance of girth exceeding w such that $I \models \Pi_1^S$ and $I \not\models \Pi_2^S$. Let J be the $\mathbf{S}_E$-instance that consists of all facts $r(a_{i_1}, \dots, a_{i_k})$ such that for some fact $R_{q(x_1,\dots,x_n)}(a_1, \dots, a_n)$, we have $r(x_{i_1}, \dots, x_{i_k}) \in q$. We show that $J \models \Pi_1$ and $J \not\models \Pi_2$.

For $J \models \Pi_1$, assume to the contrary of what is to be shown that there is an extension J' of J to schema $\mathbf{S}_E \cup \mathbf{S}_{I,1}$ such that all rules of Π_1 are satisfied and $\mathsf{goal}_1 \notin J'$. Let I' be the corresponding extension of I to $\mathbf{S}'_E \cup \mathbf{S}_{I,1}$. It suffices to show that all rules of Π_1^S are satisfied in I' to obtain a contradiction against $I \models \Pi_1^S$. Thus, let $p(\mathbf{y}) \leftarrow q(\mathbf{x})$ be a rule in Π_1^S and let h be a homomorphism from $q(\mathbf{x})$ to I'. Then there is a rule $p(\mathbf{y}) \leftarrow q'(\mathbf{x}')$ in Π_1, a relation $R_{q''(\mathbf{x}'')}$ in $\mathbf{S}'_E$, and an injective homomorphism g from $q'(\mathbf{x}')$ to $q''(\mathbf{x}'')$ such that $q(\mathbf{x}) = g^{-1}(R_{q''(\mathbf{x}'')}(\mathbf{x}'')) \wedge q'(\mathbf{x}')|_{\mathbf{S}_{I_2}}$. By construction of J and I', h is also a homomorphism from $q'(\mathbf{x}')$ to I. Since J' satisfies $p(\mathbf{y}) \leftarrow q'(\mathbf{x}')$, one of the disjuncts of $p(\mathbf{y})$ is satisfied under h, as required.

For $J \not\models \Pi_2$, let I' be an extension of I to $\mathbf{S}'_E \cup \mathbf{S}_{I,2}$ that satisfies all rules of Π_2^S and does not contain the goal_2 relation. Let J' be the corresponding extension of J. It suffices to show that J' satisfies all rules of Π_2. Thus, let $p(\mathbf{y}) \leftarrow q(\mathbf{x})$ be a rule in Π_2 and let h be a homomorphism from $q(\mathbf{x})$ to J'. Let the query $q'(\mathbf{x}')$ be obtained from $q(\mathbf{x})$ by identifying all variables that h maps to the same target. For simplicity, let us assume first that q' is connected.

Partition the EDB atoms of $q'(\mathbf{x}')$ into components as follows: every reflexive atom $r(x, \dots, x)$ forms a component and every maximal biconnected set of non-reflexive atoms forms a component. Let $q_1(\mathbf{x}_1), \dots, q_n(\mathbf{x}_n)$ be the components obtained in this way, enriched with IDB atoms in the following way: if $P(x)$ is in $q(\mathbf{x})$ with P IDB and x occurs in $q_i(\mathbf{x}_i)$, then $q_i(\mathbf{x}_i)$ contains $P(x)$. It can be verified that distinct components share at most one variable and that the undirected graph obtained by taking the non-reflexive components as nodes and putting edges between components that share a variable is a tree. For ech tree in the tree, choose a component that is the root to turn the undirected tree into a directed one, allowing us to speak about successors, predecessors, etc. Slightly extend the tree by adding each reflexive component as a leaf below some node that contains the variable in the reflexive component; if there is no such node, the component forms an extra tree.

For every component $q_i(\mathbf{x}_i)$, Π_2 contains an associated rule. Recall that we assume Π_2 to be the result of the first step of the construction of Π_2^S. If the $q_i(\mathbf{x}_i)$ is a leaf in the tree, then the associated rule takes the form $p_i(\mathbf{y}_i) \vee Q_i(x_i) \leftarrow q_i(\mathbf{x}_i)$ where $p_i(\mathbf{y}_i)$ is the restriction of $p(\mathbf{y})$ to atoms that are nullary or contain a variable from $\mathbf{x}_i$, Q_i is a fresh unary relation, and x_i is the variable that the component shares with the component which is its predecessor in the tree. For non-leafs, the rule body is additionally enriched with atoms $Q_j(x)$ where Q_j is a fresh IDB introduced for a successor node. For the root node, no fresh IDB relation is introduced.

We now make a bottom-up pass over the tree as follows. Consider the rule $p'_i(\mathbf{y}_i) \leftarrow q'_i(\mathbf{x}_i)$ associated with the current node. The homomorphism h from above is also a homomorphism from $q'_i(\mathbf{x}_i)$ to J'; this is clear for leaf nodes and can inductively be verified for inner nodes. Take the corresponding rule ρ in Π_2^S, the one that introduces a new EDB relation for the EDB atoms in $q'_i(\mathbf{x}_i)$. By construction of J', since $q'_i(\mathbf{x}_i)$ is biconnected without reflexive loops or a single reflexive loop and because the girth of I' is higher than that of $q'_i(\mathbf{x}_i)$, the h-image of all EDB-atoms in $q'_i(\mathbf{x}_i)$ must have been derived from a single fact in I. There is another rule ρ' in Π_2^S in which the EDB-relation in the body of ρ is replaced with the relation from that fact. This rule applies in

I' and thus one of the atoms from its head is true. If this is an atom from $p(\mathbf{x})$, we are done with the entire proof. If this is a fresh IDB atom, we are done with this tree node and can continue in our bottom-up tree walk. We will be done at the root at latest since the associated rule contains no fresh IDB relation.

This finishes the case where the query $q'(\mathbf{x}')$ is connected. the general case, the tree has to be replaced by a forest. There will be exactly one tree in the forest in which the rule associated with the root does not have a fresh IDB relation in the head. For those trees where the root rule has a fresh such relation, we find another tree where that relation occurs in the body of the rule associated with a leaf. This defines an order on the trees, which is acyclic. We process the trees in this order, each single tree essentially as in the connected case. ❏

From Containment to Relativized Emptiness

Lemma 23. *For any $g > 0$,*

1. *if $\Pi_1 \not\subseteq \Pi_2$, then Π is non-empty w.r.t. D.*
2. *if Π is non-empty w.r.t. D on instances of girth $> g$, for any $g > 0$, then $\Pi_1 \not\subseteq_{>g} \Pi_2$.*

Proof. For Point 1, assume $\Pi_1 \not\subseteq \Pi_2$. Then there is an $\mathbf{S}_E$-instance I such that $I \models \Pi_1$ and $I \not\models \Pi_2$. Let J be an extension of I to signature $\mathbf{S}_E \cup \mathbf{S}_{I,2}$ such that all rules in Π_2 are satisfied and $\mathsf{goal}_2() \notin J$. Add $\overline{P}(a)$ to J if $P(a) \notin J$ for all unary $P \in \mathbf{S}_{I,2}$ and $a \in \mathsf{adom}(J)$, and add $\overline{P}()$ to J if $P() \notin J$ for all nullary $P \in \mathbf{S}_{I,2}$. Clearly, (the extended) J is over schema $\mathbf{S}'_E$ and satisfies D. To show that Π is non-empty w.r.t. D, it thus remains to argue that $J \models \Pi$.

Assume to the contrary that this is not the case. Then there is an extension J' of J to signature $\mathbf{S}'_E \cup \mathbf{S}_{I,1}$ such that all rules of Π are satisfied, but $\mathsf{goal}_1() \notin J'$. Since the restriction of J' to $\mathbf{S}_E$-facts is I, it remains to argue that J' satisfies all rules in Π_1. Let $\rho p(\mathbf{y}) \longleftarrow q(\mathbf{x})$ be such a rule and let h be a homomorphism from q to J'. We define an extension q' of q as follows: conjunctively add to q all $P() \in J \cap \mathbf{S}_{I,2}$ and all $\overline{P}() \in J$ with $P \in \mathbf{S}_{I,2}$; moreover, for all variables x in q and all facts $P(h(x)) \in J$ with $P \in \mathbf{S}_{I,2}$, conjunctively add the atom $P(x)$ to q, and likewise for facts $\overline{P}(h(x))$ and atoms $\overline{P}(x)$. Since J' satisfies all rules in Π_2, it can be verified that the rule $p(\mathbf{y}) \leftarrow q'(\mathbf{x})$, which is added to Π in the first step of its construction, is not removed in the second step of the construction. Since h is a homomorphism from q' to J' and J' satisfies all rules in Π, $J' \models p[h(\mathbf{y})]$ and thus J' satifies ρ as required.

Now for Point 2. Assume that Π is non-empty w.r.t. D on instances of girth $> g$, with $g > 0$. Then there is an $\mathbf{S}'_E$-instance I of girth $> g$ with $I \models \Pi$ and $I \models D$. By construction of Π, this implies that (i) $P() \in I$ or $\overline{P}() \in I$ for every nullary $P \in \mathbf{S}_{I,2}$ (otherwise no rule of Π would be applicable, implying $I \not\models \Pi$). Moreover, we can assume w.l.o.g. that (ii) every $a \in \mathsf{adom}(I)$ satisfies $P(a) \in I$ or $\overline{P}(a) \in I$ for every unary $P \in \mathbf{S}_{I,2}$; in fact, it this is not the case, then we can simply replace I with its restriction to those elements a that satisfy the condition (elements which do not can never be involved in rule applications). Let J be the restriction of I to schema $\mathbf{S}_E$. By construction of Π and due to Conditions (i) and (ii), $I \models \Pi$ implies $J \models \Pi_1$. To finish the proof, it would this be sufficient to show that I witnesses $J \not\models \Pi_2$. This, however, need not be the case: while we know that $\mathsf{goal}_2() \in I$ as otherwise no rule of Π would be applicable, it need not be the case that all rules in Π_2 are satisfied in I. We thus show how to first manipulate I such that $I \models \Pi$ still holds, I does still not contain $\mathsf{goal}_2()$, and I satisfies all rules in Π_2. In fact, we exhaustively apply the following.

Assume that there is a rule ρ in Π_2 that is not satisfied in I. Since Π_2 is simple, ρ has the form $Q_1(y_1) \vee \cdots \vee Q_\ell(y_\ell) \leftarrow A(\mathbf{x}) \wedge P_1(\mathbf{x}_1) \wedge \cdots \wedge P_k(\mathbf{x}_k)$ with A EDB and all P_i and Q_i IDB. Let h be a homomorphism from the rule body to I such that $Q_i(h(y_i)) \notin I$ for $1 \leq i \leq \ell$. We modify I by removing the fact $A(h(\mathbf{x}))$, resulting in instance I^-. Clearly, the application of ρ via h is no longer possible and it remains to show that $I^- \models \Pi$. Assume that this is not the case, that is, there is an extension J^- of I^- to schema $\mathbf{S}'_E \cup \mathbf{S}_{I,1}$ such that $\mathsf{goal}_1() \notin J^-$ and J^- satisfies all rules of Π. Let J be the corresponding extension of I, that is, J and J^- differ only in the presence of the fact $A(h(\mathbf{x}))$. We show that J satisfies all rules of Π, contradicting $I \models \Pi$. Clearly, we need to consider only rules ρ' whose only $\mathbf{S}_E$-atom is of the form $A(\mathbf{x}')$ and only homomorphisms h' from the body of ρ' to J such that $h(\mathbf{x}') = h(\mathbf{x})$. Fix such a ρ' and h'. Since I and thus also J has girth > 1 and since $\mathbf{x}'$ contains all variables from the body of ρ', h' must be injective. By definition of Π and because of the homomorphism h', we must have

- $P() \in J$ (resp. $\overline{P}() \in J$) implies that the body of ρ' has a conjunct $P()$ (resp. $\overline{P}$) for every nullary $P \in \mathbf{S}_{I,2}$;
- $P(h'(x)) \in J$ (resp. $\overline{P}(h'(x)) \in J$) implies that the body of ρ' has a conjunct $P(x)$ (resp. $\overline{P}(x)$) for unary $P \in \mathbf{S}_{I,2}$ and all variables x from $\mathbf{x}'$.

Because of the homomorphism h and since $Q_i(h(y_i)) \notin I$ for $1 \leq i \leq \ell$, the rule $\rho \in \Pi_2$ and the variable substitution $h \circ {h'}^{-1}$ mean that the rule ρ' was removed during the second step of the construction of Π, in contradiction to $\rho' \in \Pi$. ❏

Deciding Relativized Emptiness

Lemma 9. Π *is empty w.r.t. D iff $K_\theta \not\models \Pi$ for all 0-types θ.*

Proof. Clearly, $K_\theta \models \Pi$ means that K_θ is a witness for Π being non-empty w.r.t. D. Conversely, assume that there is an $\mathbf{S}_E$-instance I with $I \models D$ and $I \models \Pi$. Let θ be the 0-type of I. Then the mapping h defined by setting $h(a) = t_a$ for all constants a in I is a homomorphism from I to K_θ. It is well-known (and can be proved using a disjunctive version of the chase procedure) that truth of MDDLog queries is preserved under homomorphisms, thus $I \models \Pi$ implies $K_\theta \models \Pi$. ❏

Lemma 10. *The algorithm for deciding relativized emptiness runs in time $\mathcal{O}(|\Pi|^3) \cdot 2^{\mathcal{O}(|D| \cdot v)}$.*

Proof. Let us first analyze the time that it takes to check whether $K_\theta \not\models \Pi$, for one instance K_θ. The number of elements in K_θ is bounded by $2^{|D|}$. Note that the interpretation of the relations in $\mathbf{S}_E \setminus \mathbf{S}_D$ is trivial, and thus we do not need to explicitly construct these relations when building K_θ. Thus, $2^{\mathcal{O}(|D|)}$ is a bound on the number of facts in (the constructed part of) K_θ and on the time needed to build it. The number of guesses to be taken when constructing K'_θ is bounded by $2^{|D|} \cdot |\mathbf{S}_I|$ where $\mathbf{S}_I$ is the IDB schema of Π, thus by $2^{|D|} \cdot |\Pi|$. For each rule in Π, the number of candidate functions for homomorphisms from the rule body to K'_θ is bounded by $2^{|D| \cdot v}$ and it can be checked in time $\mathcal{O}(|\Pi| \cdot (|D| + |\mathbf{S}_I|))$ whether a candidate is a homomorphism. We thus need time $\mathcal{O}(|\Pi|^2) \cdot 2^{\mathcal{O}(|D| \cdot v)}$ per rule and time $\mathcal{O}(|\Pi|^3) \cdot 2^{\mathcal{O}(|D| \cdot v)}$ overall to deal with the instance K_θ. There are at most $2^{|D|}$ many instances K_θ, a factor that is absorbed by the bound that we have already computed. ❑

The next lemma is a main ingredient to the proof of Lemma 11. Intuitively, it shows that a semi-simple MDDLog programs with disjointness constraints can be understood as a constraint satisfaction problem (CSP). For two instances I and J, we write $I \rightarrow J$ if there is a homomorphism from I to J, which is defined in the standard way. Of course, homomorphisms also have to respect nullary relations. In the following, we call a finite instance a *template* when we use it as a homomorphism target, as in the CSP literature.

Lemma 24. *For every 0-type θ, there are templates $T_0, \dots, T_n$ in signature $\mathbf{S}_E$ such that, for every $\mathbf{S}_E$-instance I of some 0-type $\theta' \subseteq \theta$, we have $I \not\models \Pi$ iff $I \rightarrow T_i$ for some $i \leq n$.*

Proof. Just like the construction of the instances K_θ, the construction of the templates $T_0, \dots, T_n$ is based on types. However, the types used for this purpose are formed over the schema $\mathbf{S} = \mathbf{S}_I \cup \mathbf{S}_D$ instead of over the schema $\mathbf{S}_D$, where $\mathbf{S}_I$ is the IDB schema of Π. To emphasize the difference between the two kinds of types, we from now on call the types introduced above *types for* $\mathbf{S}_D$.

A *type for* $\mathbf{S}$ is a set $t \subseteq \mathbf{S}$ that satisfies all rules in Π, that is, if a rule ρ mentions only nullary and unary relations (and thus involves at most a single variable since Π is semi-simple w.r.t. D) and all these relations are in t, then at least one of the relations from the head of ρ is in t. Note that, in contrast to 0-types for $\mathbf{S}_D$ and 1-types for $\mathbf{S}_D$ defined before, a type for $\mathbf{S}$ contains both unary and nullary relation symbols. The restriction δ of a type t for $\mathbf{S}$ to nullary relations is a *0-type for* $\mathbf{S}$. There is no need to define 1-types for $\mathbf{S}$. We say that a type t for $\mathbf{S}$ is *compatible* with a 0-type δ for $\mathbf{S}$ if δ is the restriction of t to nullary relations.

Let θ be a 0-type for D. To construct the set of templates stipulated in Lemma 24, we define one template T_δ for every 0-type δ for $\mathbf{S}$ that agrees with θ on relations from $\mathbf{S}_D$ and does not contain the goal relation. The elements of T_δ are the types for $\mathbf{S}$ that are compatible with δ and T_δ consists of the following facts:

1. all facts $R(t_1, \dots, t_n)$ such that $R \in \mathbf{S}_E \setminus \mathbf{S}_D$ is of arity n and there is no rule ρ in Π and variables $x_1, \dots, x_n$ such that the following conditions are satisfied:
   - $R(x_1, \dots, x_n)$ occurs in the body of ρ;
   - if $P(x_i)$ or $P()$ occurs in the body of ρ with $P \in \mathbf{S}$, then $P \in t_i$;
   - for none of the disjuncts $P(x_i)$ in the head of ρ, we have $P \in t_i$;
   - for none of the disjuncts $P()$ in the head of ρ, we have $P \in \delta$;
2. all facts $P(t)$ with $P \in t \cap \mathbf{S}_D$;
3. all facts $P()$ with $P \in \delta \cap \mathbf{S}_D$.

We have to show that the templates T_δ are as required, that is, for every $\mathbf{S}_E$-instance I that has some 0-type $\theta' \subseteq \theta$ for $\mathbf{S}_D$, we have $I \not\models \Pi$ iff $I \rightarrow T_\delta$ for some 0-type δ for $\mathbf{S}$.

"if". Assume that I is an $\mathbf{S}_E$-instance of 0-type $\theta' \subseteq \theta$ for $\mathbf{S}_D$ and that there is a homomorphism h from I to T_δ. Extend I to an $\mathbf{S}_E \cup \mathbf{S}_I$-instance J by adding $P()$ whenever $P() \in \delta \cap \mathbf{S}_I$ and $P(a)$ whenever $P \in h(a) \cap \mathbf{S}_I$. Since the goal relation is not in δ, it is not in J; it thus remains to show that every rule ρ of Π is satisfied in J. First for rules ρ that contain a body atom whose relation is from $\mathbf{S}_E \setminus \mathbf{S}_D$. Since Π is semi-simple w.r.t. D, the body of ρ consists of one atom $R(x_1, \dots, x_n)$ with $R \in \mathbf{S}_E \setminus \mathbf{S}_D$ plus atoms of the form $P(x_i)$ and $P()$ with $P \in \mathbf{S}$. Assume that g is a homomorphism from the body of ρ to J. Then we have $R(g(x_1), \dots, g(x_n)) \in T_\delta$. Moreover,

1. when $P(x_i)$ is in the body of ρ and $P \in \mathbf{S}$, then $P \in g(x_i)$ and
2. when $P()$ is in the body of ρ and $P \in \mathbf{S}$, then $P \in g(x_i)$ for $1 \leq i \leq n$.

In fact, Point 1 follows from Point 2 of the construction of T_δ when $P \in \mathbf{S}_D$ and from the definition of J when $P \in \mathbf{S}_I$. Point 2 similarly follows from Point 3 of the construction of T_δ. With Points 1 and 2 above, Condition 1 from the construction of T_δ yields that the head of ρ contains an atom $P(x_i)$ with $P \in g(x_i)$ or $P()$ with $P \in J$. Thus, ρ is satisfied in J. For rules ρ that contain no atom with a relation from $\mathbf{S}_E \setminus \mathbf{S}_D$, one can first observe that, since Π is semi-simple w.r.t. D, there is at most one variable in the rule. We can now argue in a similar way as before that the rule is satisfied in J, using in particular the fact that types for $\mathbf{S}$ satisfy the rules in Π.

"only if". Assume that I is an $\mathbf{S}_E$-instance of 0-type $\theta' \subseteq \theta$ for $\mathbf{S}_D$ and that $I \not\models \Pi$. Then there is an extension J of I to signature $\mathbf{S}_E \cup \mathbf{S}_I$ that satisfies rules of Π and does not contain the goal relation. For every constant a in I, let t_a be the set of all unary relations $P \in \mathbf{S}$ such that $P(a) \in J$ and of all nullary relations $P \in \mathbf{S}$ such that $P() \in J$. Clearly, t_a is a type for $\mathbf{S}$. Set $h(a) = t_a$ for all constants a in I. It remains to verify that h is a homomorphism from I to T_δ.

First consider facts $P(a) \in I$ with $P \in \mathbf{S}_D$. Then $P \in t_a$, thus $P \in h(a)$, thus $P(a) \in T_\delta$ by definition of T_δ.

Now consider facts $R(a_1, \dots, a_n) \in I$ with $R \in \mathbf{S}_E \setminus \mathbf{S}_D$. Assume to the contrary of what is to be shown that $R(h(a_1), \dots, h(a_n)) \notin T_\delta$. By definition of T_δ, there is a rule ρ of Π and variables $x_1, \dots, x_n$ such that

1. $R(x_1, \dots, x_n)$ occurs in the body of ρ,
2. if $P(x_i)$ or $P()$ occurs in the body of ρ with $P \in \mathbf{S}$, then $P \in t_i$;
3. for none of the disjuncts $P(x_i)$ in the head of ρ, we have $P \in t_i$;
4. for none of the disjuncts $P()$ in the head of ρ, we have $P \in \delta$.

By definition of T_δ and of h, Points 2 to 4 imply that

5. if $P(x_i)$ (resp $P()$) occurs in the body of ρ with $P \in \mathbf{S}$, then $P(a_i) \in J$ (resp. $P() \in J$);
6. for none of the disjuncts $P(x_i)$ in the head of ρ, we have $P(a_i) \in J$;
7. for none of the disjuncts $P()$ in the head of ρ, we have $P() \in J$;

Since $R(x_1, \dots, x_n)$ occurs in the body of ρ and Π is semi-simple w.r.t. D, the variables $x_1, \dots, x_n$ are all distinct. We can thus define a function g by setting $g(x_i) = a_i$. Since $R(a_1, \dots, a_n) \in I$ and by Point 5, g is a homomorphism from the body of ρ to I. By Points 6 and 7, g witnesses that ρ is violated in J, in contradiction to J satisfying all rules in Π. ❏

A second important ingredient to the proof of Theorem 11 is the following well-known lemma which is originally due to Erdős and concerns graphs of high girth and high chromatic number and was adapted to the following formulation in (Feder and Vardi 1998).

Lemma 25. *Let $\mathbf{S}_E$ be a schema. For every $\mathbf{S}_E$-instance I and $g, s \geq 0$, there is an $\mathbf{S}_E$-instance I' such that*

1. *$I' \rightarrow I$,*
2. *I' has girth exceeding g, and*
3. *for every $\mathbf{S}_E$-instance J with at most s elements, $J \rightarrow I$ iff $J \rightarrow I'$.*

Lemma 11. *For every Boolean MDDLog program Π over EDB schema $\mathbf{S}_E$ and set of disjointness constraints D over $\mathbf{S}_E$ such that Π is semi-simple w.r.t. D, the following are equivalent for any $g \geq 0$:*

1. *Π is empty regarding D and*
2. *Π is empty regarding D and instances of girth exceeding g.*

Proof. Clearly, we only need to prove that if Π is non-empty w.r.t. D, then this is witnessed by an instance of girth exceeding g. By Lemma 9, Π being non-empty w.r.t. D implies that $K_\theta \models \Pi$ for some 0-type θ. By Lemma 24, we find templates $T_0, \dots, T_n$ in signature $\mathbf{S}_E$ such that, for every $\mathbf{S}_E$-instance I of some 0-type $\theta' \subseteq \theta$, we have $I \not\models \Pi$ iff $I \rightarrow T_i$ for some $i \leq n$. Thus $K_\theta \not\rightarrow T_i$ for all $i \leq n$. By Lemma 25, there is an $\mathbf{S}_E$-instance K'_θ such that $K'_\theta \rightarrow K_\theta$ (and thus K'_θ satisfies the constraints in D and has some 0-type $\theta' \subseteq \theta$), K'_θ has girth exceeding h, and for every $\mathbf{S}_E$-instance I of size at most $s := \max\{|T_0|, \dots, |T_n|\}$, we have $K'_\theta \rightarrow I$ iff $K_\theta \rightarrow I$. The latter implies $K'_\theta \not\rightarrow T_i$ for all $i \leq n$, and thus $K'_\theta \models \Pi$ as desired. □ ❏

Deriving Upper Bounds

Theorem 12. *Containment between Boolean MDDLog programs and between MMSNP sentences is in* 2NExpTime. *More precisely, for Boolean MDDLog programs Π_1 and Π_2, it can be decided non-deterministically in time $2^{2^{p(|\Pi_2| \cdot \log |\Pi_1|)}}$ whether $\Pi_1 \subseteq \Pi_2$, p a polynomial.*

Proof. To decide whether $\Pi_1 \subseteq \Pi_2$, we first jointly simplify Π_1 and Π_2 as per Theorem 6, giving programs Π_1^S and Π_2^S. These programs and Theorem 7 give another program Π and a set of constraints D such that Π is semi-simple w.r.t. Π. We decide whether Π is empty w.r.t. D and return the result. The size and complexity bounds given in Theorems 6, 7, and 8 give the complexity bound stated in Theorem 12. In fact, it can be verified that $|\Pi_i^S| \leq 2^{p(|\Pi_i|)}$, $|\Pi| \leq 2^{2^{p(|\Pi_2| \cdot \log |\Pi_1|)}}$, and $|D| \leq 2^{p(\Pi_2)}$ where p is a polynomial. Moreover, the variable width of Π is bounded by that of $\Pi_1 \cup \Pi_2$ and it remains to plug these bounds into the time bounds stated in Theorem 8.

It remains to argue that the algorithm is correct. If $\Pi_1 \not\subseteq \Pi_2$, then $\Pi_1^S \not\subseteq \Pi_2^S$, thus Π is non-empty w.r.t. D, thus "no" is returned. Let w be the atom width of $\Pi_1 \cup \Pi_2$ (with the exception that $w = 2$ if that atom width is one). If "no" is returned by our algorithm, then Π is non-empty w.r.t. D. By Theorem 11, Π is non-empty w.r.t. D on instances of girth $> w$. Thus $\Pi_1^S \not\subseteq_{>w} \Pi_2^S$, implying $\Pi_1 \not\subseteq \Pi_2$. ❏

For the following lemma, let $\mathbf{S}_E$ be the EDB schema of Π_1 and Π_2, and let $\mathbf{S}_{I,i}$ be the IDB schema of Π_i, $i \in \{1, 2\}$.

Lemma 13. $\Pi_1 \subseteq \Pi_2$ *iff* $\Pi_1^{\mathbf{a}} \subseteq \Pi_2^{\mathbf{a}}$ *for all* $\mathbf{a} \in \mathbf{C}^k$.

Proof. Assume that $\Pi_1 \not\subseteq \Pi_2$. Then there is an $\mathbf{S}_E$-instance I and a tuple $\mathbf{a} \subseteq \mathsf{adom}(I)^k$ such that $\mathbf{a} \in \Pi_1(I) \setminus \Pi_2(I)$. Let $\mathbf{C}_\Pi$ be the constants which occur in $\Pi_1 \cup \Pi_2$, and observe that $\mathbf{C}_\Pi \subseteq \mathbf{C}$. We can assume w.l.o.g. that all constants in $\mathbf{a}$ are from $\mathbf{C}$; if this is not the case, we can first rename constants in I and $\mathbf{a}$ that are from $\mathbf{C} \setminus (\mathbf{C}_\Pi \cup \mathbf{a})$ with fresh constants and then rename constants in I and $\mathbf{a}$ that are from $\mathbf{a} \setminus \mathbf{C}$ with constants from $\mathbf{C} \setminus \mathbf{C}_\Pi$. By choice of $\mathbf{C}$, there are enough constants of the latter kind. It can now be verified that $I \models \Pi_1^{\mathbf{a}}$ and $I \not\models \Pi_2^{\mathbf{a}}$. In particular, an extension J of I to schema $\mathbf{S}_E \cup \mathbf{S}_{I,1}$ that satisfies all rules in $\Pi_1^{\mathbf{a}}$ and does not contain the goal relation gives rise to an extension J' of I that satisfies all rules in $\Pi_1^{\mathbf{a}}$ and does not contain $\mathsf{goal}(\mathbf{a})$: start with J' and then apply all goal rules of Π_1. By construction of $\Pi_1^{\mathbf{a}}$ and since $\mathsf{goal}() \notin J'$, we have $\mathsf{goal}(\mathbf{a}) \notin J'$. Similarly, an extension J of I to schema $\mathbf{S}_E \cup \mathbf{S}_{I,2}$ that satisfies all rules in Π_2 and does not contain $\mathsf{goal}(\mathbf{a})$ gives rise to an extension J' of I that satisfies all rules in $\Pi_2^{\mathbf{a}}$ and does not contain $\mathsf{goal}()$.

Conversely, assume that $\Pi_1^{\mathbf{a}} \not\subseteq \Pi_2^{\mathbf{a}}$. Then there is an $\mathbf{S}_E$-instance I with $I \models \Pi_1^{\mathbf{a}}$ and $I \not\models \Pi_1^{\mathbf{a}}$. By considering appropriate extensions of I to the schemas $\mathbf{S}_E \cup \mathbf{S}_{I,i}$, it can be verified in a very similar way as above that $\mathbf{a} \in \Pi_1(I) \setminus \Pi_2(I)$. ❏

Lemma 14. $\Pi_1 \subseteq \Pi_2$ *iff* $\Pi'_1 \subseteq \Pi'_2$.

Proof. Assume that $\Pi_1 \not\subseteq \Pi_2$. Then there is an $\mathbf{S}_E$-instance I such that $I \models \Pi_1$ and $I \not\models \Pi_2$. Let J be the $\mathbf{S}'_E$-instance obtained from I by adding $R_a(a)$ whenever $a \in \mathsf{adom}(I) \cap \mathbf{C}$. It can be verified that $J \models \Pi'_1$ and $J \not\models \Pi'_2$.

Conversely, assume that $\Pi'_1 \not\subseteq \Pi'_2$. Then there is an $\mathbf{S}'_E$-instance I such that $I \models \Pi'_1$ and $I \not\models \Pi'_2$. Let J be the $\mathbf{S}_E$-instance which is obtained from I by

- dropping all facts that use a relation from $\mathbf{S}'_E \setminus \mathbf{S}_E$ and then
- taking the quotient according to the following equivalence relation on $\mathsf{adom}(I)$:
$$\{(a,b) \mid \exists c \in \mathbf{C} : R_c(a), R_c(b) \in I\};$$
note that this is indeed an equivalence relation because the rules $\mathsf{goal}() \leftarrow R_{a_1}(x) \wedge R_{a_2}(x)$ in Π'_2 (for all distinct a_1, a_2) imply that for any $a \in \mathsf{adom}(I)$, there is at most one $b \in \mathbf{C}$ with $R_b(a) \in I$.

It can be verified that $J \models \Pi_1$ and $J \not\models \Pi_2$. ❏

Ontology-Mediated Queries

Preliminaries We first introduce the relevant OMQ languages and then provide missing proofs.

A $\mathcal{ALCI}$*-concept* is formed according to the syntax rule

$$C, D ::= \top \mid \bot \mid A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \\ \exists r.C \mid \exists r^-.C \mid \forall r.C \mid \forall r^-.C$$

where A ranges over a fixed countably infinite set of *concept names* and r over a fixed countably infinite set of *role names*. An $\mathcal{ALC}$*-concept* is an $\mathcal{ALCI}$-concept in which the constructors $\exists r^-.C$ and $\forall r^-.C$ are not used. An $\mathcal{ALC}$-TBox (resp. $\mathcal{ALCI}$-TBox) is a finite set of concept inclusions $C \sqsubseteq D$, C and D $\mathcal{ALC}$-concepts (resp. $\mathcal{ALCI}$-concepts). A $\mathcal{SHI}$*-TBox* is a finite set of

- *concept inclusions* $C \sqsubseteq D$, C and D $\mathcal{SHI}$-concepts,
- *role inclusion* $r \sqsubseteq s$, r and s role names, and
- *transitivity statements* $\mathsf{trans}(r)$, r a role name.

DL semantics is given in terms of interpretations. An *interpretation* takes that form $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ where $\Delta^\mathcal{I}$ is a non-empty set called the *domain* and $\cdot^\mathcal{I}$ is the *interpretation function* which maps each concept name A to a subset $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$ and each role name r to a binary relation $r^\mathcal{I} \subseteq r^\mathcal{I} \times r^\mathcal{I}$. The interpretation functions is extended to concepts in the standard way, for example

$$(\exists r.C)^\mathcal{I} = \{d \in \Delta^\mathcal{I} \mid \exists e \in C^\mathcal{I} : (d,e) \in r^\mathcal{I}\}$$
$$(\exists r^-.C)^\mathcal{I} = \{d \in \Delta^\mathcal{I} \mid \exists e \in C^\mathcal{I} : (e,d) \in r^\mathcal{I}\}.$$

We refer to standard references such as (**?**) for full details. An intepretation is a *model* of a TBox $\mathcal{T}$ if it *satisfies* all statements in $\mathcal{T}$, that is,

- $C \sqsubseteq D \in \mathcal{T}$ implies $C^\mathcal{I} \subseteq D^\mathcal{I}$;
- $r \sqsubseteq s \in \mathcal{T}$ implies $r^\mathcal{I} \subseteq s^\mathcal{I}$;
- $\mathsf{trans}(r) \in \mathcal{T}$ implies that $r^\mathcal{I}$ is transitive.

In description logic, data is typically stored in so-called ABoxes. For uniformity with MDDLog, we use instances instead, identifying unary relations with concept names, binary relations with role names, and disallowing relations of any other arity. An interpretation $\mathcal{I}$ is a *model* of an instance I if $A(a) \in I$ implies $a \in A^\mathcal{I}$ and $r(a,b) \in I$ implies $(a,b) \in r^\mathcal{I}$. We say that an instance I is *consistent* with a TBox $\mathcal{T}$ if I and $\mathcal{T}$ have a joint model. We write $\mathcal{T} \models r \sqsubseteq s$ if every model $\mathcal{I}$ of $\mathcal{T}$ satisfies $r^\mathcal{I} \subseteq s^\mathcal{I}$.

An *ontology-mediated query (OMQ)* takes the form $Q = (\mathcal{T}, \mathbf{S}_E, q)$ with $\mathcal{T}$ a TBox, $\mathbf{S}_E$ a set of concept and role names, and q a UCQ. We use $(\mathcal{L}, \mathcal{Q})$ to refer to the set of all OMQs whose TBox is formulated in the language $\mathcal{L}$ and where the actual queries are from the language $\mathcal{Q}$. For example, $(\mathcal{ALC}, \text{UCQ})$ refers to the set of all OMQs that consist of an $\mathcal{ALC}$-TBox and a UCQ. For OMQs $(\mathcal{T}, q)$ from $(\mathcal{SHI}, \mathcal{UCQ})$, we adopt the following additional restriction: when $\mathcal{T}$ contains a transitivity $\mathsf{trans}(r)$ and $\mathcal{T} \models r \sqsubseteq s$, we disallow the use of s in the query q. Let I be an $\mathbf{S}_E$-instance and $\mathbf{a}$ a tuple of constants from I. We write $I \models Q[\mathbf{a}]$ and call $\mathbf{a}$ a *certain answer to Q on I* if for all models $\mathcal{I}$ of I and $\mathcal{T}$, we have $\mathcal{I} \models q[\mathbf{a}]$ (defined in the usual way).

Containment between OMQs is defined in analogy with containment between MDDLog programs: $Q_1 = (\mathcal{T}_1, \mathbf{S}_E, q_1)$ is *contained* in $Q_2 = (\mathcal{T}_2, \mathbf{S}_E, q_2)$, written $Q_1 \subseteq Q_2$, if for every $\mathbf{S}_E$-instance I and tuple $\mathbf{a}$ of constants from I, $I \models Q_1[\mathbf{a}]$ implies $I \models Q_2[\mathbf{a}]$. This is different from the notion of containment considered in (Bienvenu, Lutz, and Wolter 2012), here called *consistent containment*. We say that $Q_1 = (\mathcal{T}_1, \mathbf{S}_E, q_1)$ is *consistently contained* in $Q_2 = (\mathcal{T}_2, \mathbf{S}_E, q_2)$, written $Q_1 \subseteq^\mathbf{c} Q_2$, if for every $\mathbf{S}_E$-instance I that is consistent with $\mathcal{T}_1$ and $\mathcal{T}_2$ and every tuple $\mathbf{a}$ of constants from I, $I \models Q_1[\mathbf{a}]$ implies $I \models Q_2[\mathbf{a}]$. We observe the following.

Lemma 26. *In $(\mathcal{SHI}, UCQ)$, consistent containment can be reduced to containment in polynomial time.*

Proof.(sketch) Let $Q_1 = (\mathcal{T}_1, \mathbf{S}_E, q_1)$ and $Q_2 = (\mathcal{T}_2, \mathbf{S}_E, q_2)$ be OMQs from $(\mathcal{SHI}, \text{UCQ})$. Assume without loss of generality that all concept that occur in $\mathcal{T}_1$ are in negation normal form, that is, negation is only applied to concept names but not to compound concepts. For every concept name A in $\mathbf{S}_E \cup \mathsf{sig}(\mathcal{T}_1)$, introduce fresh concept names A' and $\overline{A}'$ that do not occur in Q_1 and Q_2. For every role name r in $\mathbf{S}_E \cup \mathsf{sig}(\mathcal{T}_1)$, introduce a fresh role name r'. Define the TBox $\mathcal{T}'_2$ as the extension of $\mathcal{T}_2$ with the following:

- $A \sqsubseteq A'$ for all concept names A in $\mathbf{S}_E \cup \mathsf{sig}(\mathcal{T}_1)$;
- $r \sqsubseteq r'$ for all role names r in $\mathbf{S}_E \cup \mathsf{sig}(\mathcal{T}_1)$;
- every concept inclusion, role inclusion, and transitivity statement from $\mathcal{T}_1$, each concept name A replaced with A', each subconcept $\neg A$ replaced with $\overline{A}'$, and each role name replaced with r';
- the inclusions $\top \sqsubseteq A' \sqcup \overline{A}'$ and $A' \sqcap \overline{A}' \sqsubseteq B$ for all concept names A in $\mathbf{S}_E \cup \mathsf{sig}(\mathcal{T}_1)$, where B is a fresh concept name.

Set $q_2' = q_2 \vee \exists x\, B(x)$. It suffices to establish the following claim. The proof is not difficult and left to the reader.

Claim. $Q_1 \subseteq^{\mathsf{c}} Q_2$ iff $Q_1 \subseteq Q_2'$. ❑

Upper Bound

Theorem 19. *For every OMQ $Q = (\mathcal{T}, \mathbf{S}_E, q)$ from $(\mathcal{SHI}, UCQ)$, one can construct an equivalent MDDLog program Π such that*

1. $|\Pi| \leq 2^{2^{p(|q|\cdot\log|\mathcal{T}|)}}$;

2. the IDB schema of Π is of size $2^{p(|q|\cdot\log|\mathcal{T}|)}$;

3. the rule size of Π is bounded by $|q|$

where p is a polynomial. The construction takes time polynomial in $|\Pi|$.

Proof. Let $Q = (\mathcal{T}, \mathbf{S}_E, q_0)$ be an OMQ from $(\mathcal{SHI}, \text{UCQ})$. We use $\mathsf{sub}(\mathcal{T})$ to denote the set of subconcepts of (concepts occurring in) $\mathcal{T}$. Moreover, let Γ be the set of all tree-shaped conjunctive queries that can be obtained from a CQ in q_0 by first quantifying all answer variables, then identifying variables, and then taking a subquery. Here, a conjunctive query q is *tree-shaped* if (i) the undirected graph $(V, \{\{x,y\} \mid r(x,y) \in q\})$ is a tree (where V is the set of variables in q), (ii) $r_1(x,y), r_2(x,y) \in q$ implies $r_1 = r_2$, and (iii) $r(x,y) \in q$ implies $s(y,x) \notin q$ for all s. Every $q \in \Gamma$ can be viewed as a $\mathcal{ALCI}$-concept provided that we additionally choose a root x of the tree. We denote this concept with $C_{q,x}$. For example, the query $\exists x \exists y \exists z\, r(x,y) \wedge A(y) \wedge s(x,z)$ yields the $\mathcal{ALCI}$-concept $\exists r.A \sqcap \exists s.\top$. Let $\mathsf{con}(q_0)$ be the set of all these concepts $C_{q,x}$ and let $\mathbf{S}_I$ be the schema that consists of monadic relation symbols P_C and $\overline{P}_C$ for each $C \in \mathsf{sub}(\mathcal{T}) \cup \mathsf{con}(q_0)$ and nullary relation symbols P_q and $\overline{P}_q$ for each $q \in \Gamma$. We are going to construct an MDDLog program Π over EDB schema $\mathbf{S}_E$ and IDB schema $\mathbf{S}_I$ that is equivalent to Q.

By a *diagram*, we mean a conjunction $\delta(\mathbf{x})$ of atoms over the schema $\mathbf{S}_E \cup \mathbf{S}_I$. For an interpretation $\mathcal{I}$, we write $\mathcal{I} \models \delta(\mathbf{x})$ if there is a homomorphism from $\delta(\mathbf{x})$ to $\mathcal{I}$, that is, a map $h : \mathbf{x} \to \Delta^{\mathcal{I}}$ such that:

1. $A(x) \in \delta$ with $A \in \mathbf{S}_E$ implies $h(x) \in A^{\mathcal{I}}$;
2. $r(x,y) \in \delta$ with $r \in \mathbf{S}_E$ implies $(h(x), h(y)) \in A^{\mathcal{I}}$;
3. $P_q() \in \delta$ implies $\mathcal{I} \models q$ and $\overline{P}_q() \in \delta$ implies $\mathcal{I} \not\models q$;
4. $P_C(x) \in \delta$ implies $h(x) \in C^{\mathcal{I}}$ and $\overline{P}_C() \in \delta$ implies $h(x) \notin C^{\mathcal{I}}$.

We say that $\delta(\mathbf{x})$ is *realizable* if there is an interpretation $\mathcal{I}$ with $\mathcal{I} \models \delta(\mathbf{x})$. A diagram $\delta(\mathbf{x})$ *implies* a CQ $q(\mathbf{x}')$, with $\mathbf{x}'$ a sequence of variables from x, if every homomorphism from $\delta(\mathbf{x})$ to some interpretation $\mathcal{I}$ is also a homomorphism from $q(\mathbf{x}')$ to $\mathcal{I}$. The MDDLog program Π consists of the following rules:

1. the rule $P_q() \vee \overline{P}_q() \leftarrow R(\mathbf{x})$ for each $q \in \Gamma$, each $R \in \mathbf{S}_E$, and each $R \in \mathbf{S}_E$ where $\mathbf{x} = x_1, \dots, x_n$, n the arity of R;
2. the rule $P_C(x) \vee \overline{P}_C(x) \leftarrow R(\mathbf{x})$ for each $C \in \mathsf{sub}(\mathcal{T}) \cup \mathsf{con}(q_0)$, each $R \in \mathbf{S}_E$, and each tuple $\mathbf{x}$ that can be obtained from $x_1, \dots, x_n$ by replacing a single x_i with x (n the arity of R);
3. the rule $\bot \leftarrow \delta(x)$ for each non-realizable diagram $\delta(x)$ that contains a single variable x and only atoms of the form $P_C(x)$, $C \in \mathsf{sub}(\mathcal{T}) \cup \mathsf{con}(q_0)$;
4. the rule $\bot \leftarrow \delta(\mathbf{x})$ for each non-realizable connected diagram $\delta(\mathbf{x})$ that contains at most two variables and at most three atoms;
5. the rule $\mathsf{goal}(\mathbf{x}') \leftarrow \delta(\mathbf{x})$ for each diagram $\delta(\mathbf{x})$ that implies $q_0(\mathbf{x})$, has at most $|q_0|$ variable occurrences, and uses only relations of the following form: P_q, P_C with C a concept name that occurs in q_0, and role names from $\mathbf{S}_E$ that occur in q_0.

To understand Π, a good first intuition is that rules of type 1 and 2 guess an interpretation $\mathcal{I}$, rules of type 3 and 4 take care that the independent guesses are consistent with each other, with the facts in I and with the inclusions in the TBox $\mathcal{T}$, and rules of type 5 ensure that Π returns the answers to q_0 in $\mathcal{I}$.

However, this description is an oversimplification. Guessing $\mathcal{I}$ is not really possible since $\mathcal{I}$ might have to contain additional domain elements to satisfy existential quantifiers in $\mathcal{T}$ which may be involved in homomorphisms from (a CQ in) q_0 to $\mathcal{I}$, but new elements cannot be introduced by MDDLog rules. Instead of introducing new elements, rules of type 1 and 2 thus only guess the tree-shaped queries that are satisfied by those elements. Tree-shaped queries suffice because $\mathcal{SHI}$ has a tree-like model property and since we have disallowed the use of roles in the query that have a transitive subrole. The notion of 'diagram implies query' used in the rules of type 4 takes care that the guessed tree-shaped queries are taken into account when looking for homomorphisms from q_0 to the guessed model. A more detailed explanation can be found in the proof of Theorem 1 of (Bienvenu et al. 2013). In fact, the construction used there is identical to the one used here, with two exceptions. First, we use predicates P_C and $\overline{P}_C$ for every concept $C \in \mathsf{sub}(\mathcal{T}) \cup \mathsf{con}(q_0)$ while the mentioned proof uses a predicate P_t for every subset $t \subseteq \mathsf{sub}(\mathcal{T}) \cup \mathsf{con}(q_0)$. And second, our versions of Rules 3-5 are formulated more carefully. It can be verified that the correctness proof given in (Bienvenu et al. 2013) is not affected by these modifications. The modifications do make a difference regarding the size of Π, though, which we analyse next.

It is not hard to see that the number of rules of type 1 is bounded by $2^{|q|^2}$, the number of rules of type 2 is bounded by $|\mathcal{T}|$, the number of rules of type 3 is bounded by $2^{2^{|q|\cdot\log|\mathcal{T}|}}$, the number of rules of type 4 is bounded by $2^{p(|q|\cdot\log|\mathcal{T}|)}$ for some polynomial p, and the number of rules of type 5 is bounded by $2^{p(|q|)}$. Consequently, the overall number of rules is bounded by $2^{2^{p(|q|\cdot\log|\mathcal{T}|)}}$ and so is the size of Π. The bounds on the size of the IDB schema and number of rules in Π stated in Theorem 19 are easily verified. The construction can be carried out in double exponential time since for a given diagram $\delta(\mathbf{x})$ and CQ $q(\mathbf{x}')$, with $\mathbf{x}'$ a sequence of variables from x, it can be decided in 2EXPTIME whether $\delta(\mathbf{x})$ implies $q(\mathbf{x}')$. ❑

Theorem 20. *Containment between OMQs from $(\mathcal{SHI}, UCQ)$ is in* 2NEXPTIME. *More precisely, for OMQs $Q_1 = (\mathcal{T}_1, \mathbf{S}_E, q_1)$ and $Q_2 = (\mathcal{T}_2, \mathbf{S}_E, q_2)$, it can be decided non-deterministically in time $2^{2^{p(|q_1|\cdot|q_2|\cdot\log|\mathcal{T}_1|\cdot\log|\mathcal{T}_2|)}}$ whether $Q_1 \subseteq Q_2$, p a polynomial.*

Proof. We convert Q_1 and Q_2 into MDDLog programs as per Theorem 19 and then remove the answer variables according to the proof of Theorem 15. Analyzing the latter construction reveals that it produces programs of size $r \cdot 2s \cdot a^s$ where r is the number of rules of the input program, s is the rule size, and a the arity. Moreover the IDB schema is not changed and rule size at most doubles. The Π_1, Π_2 obtained by these two first steps thus still satisfy Conditions 1-3 of Theorem 19 except that $|q|$ in the last point has to be replaced by $2|q|$.

The joint simplifications Π_1^S and Π_2^S from Theorem 6 then have size $|\Pi_i^S| \leq 2^{2^{p(|q_i|\cdot\log|\mathcal{T}_i|)}}$ and their variable width is bounded by (the rule size of Π_i and thus by) $2|q|$. Let us analyze the size of the IDB schema of Π_i^S. First note that the initial variable identification step can be ignored. In fact, we start with at most $2^{2^{p(|q_i|\cdot\log|\mathcal{T}_i|)}}$ rules, each of size at most $2|q_i|$. Thus variable identification results in a factor of $(2|q_i|)!$, which is absorbed by $2^{2^{p(|q_i|\cdot\log|\mathcal{T}|)}}$. The other parameters are not changed by variable identification.

When making rules biconnected in the construction of Π_1^S and Π_2^S, we need not worry about rules of type 1-2 and 4-5. The reason is that there are only $2^{p(|q_i|\cdot\log|\mathcal{T}_i|)}$ many such rules, each of size at most $2|q_i|$, and thus the number of additional IDB relations introduced for making them biconnected is also bounded by $2^{p(|q_i|\cdot\log|\mathcal{T}_i|)}$. Rules of type 3, on the other hand, are of a very restricted form, namely

$$\bot \leftarrow P_{C_1}(x) \wedge \cdots \wedge P_{C_n}(x)$$

with $C_1, \ldots, C_n \in \mathsf{sub}(\mathcal{T}) \cup \mathsf{con}(q_0)$. These rules are biconnected and thus we are done in the Boolean case. In the non-Boolean case, rules of the above form are manipulated in the second step of the reduction of non-Boolean MDDLog programs to Boolean MDDLog programs. The result are rules of exactly the same shape, but also rules of the form

$$\bot \leftarrow P_{C_1}(x_1) \wedge R_a(x_1) \wedge \cdots \wedge P_{C_n}(x_n) \wedge R_a(x_n).$$

The latter rules have to be split up to be made biconnected. This will result in rules of the form

$$\bot \leftarrow Q_1() \wedge \cdots \wedge Q_n() \quad \text{and} \quad Q_i() \leftarrow P_C(x) \wedge R_a(x)$$

where R_a is one of the fresh IDB relations introduced in the mentioned reduction. Clearly, there are only $2^{p(|q_i|\cdot\log|\mathcal{T}_i|)}$ many rule bodies of the latter form and thus it suffices to introduce at most the same number of fresh IDB relations Q_i. In summary, we have shown that a careful construction of Π_i^S can ensure that the size of the IDB schema of Π_i^S is bounded by $2^{p(|q_i|\cdot\log|\mathcal{T}_i|)}$.

It can now be verified that the program Π from Theorem 7 has size at most $2^{2^{p(|q_1|\cdot|q_2|\cdot\log|\mathcal{T}_1|\cdot\log|\mathcal{T}_2|)}}$ and D has size $2^{p(|q_2|\cdot\log|\mathcal{T}_2|)}$. Applying Theorem 8 gives the complexity bound stated in Theorem 20. ❑

Lower Bound The following result establishes the lower bound in Point 3 of Theorem 17. We state it here even in a slightly stronger form. $\mathcal{ELIU}$ denotes the description logic that admits only the concept constructors $\top$, $\sqcap$, $\sqcup$, and $\exists$ and $\mathcal{ELI}_\bot$ denotes the DL with the constructors $\top$, $\bot$, $\sqcap$, and $\exists$. With BAQ, we denote the class of *Boolean atomic queries*, that is, queries of the form $\exists x\, A(x)$ with A a concept name.

Theorem 27. *Containment of an $(\mathcal{ELIU}, BAQ)$-OMQ in an $(\mathcal{ELI}_\bot, CQ)$-OMQ is* 2NEXPTIME*-hard.*

The overall strategy of the proof is similar to that of our proof of the lower bounds stated in Theorem 3, but the details differ in a number of respects. Instead of reducing 2-exp square tiling problem, we now reduce the 2-exp torus tiling problem. The definition is identical except that a tiling f for the latter problem additionally needs to satisfy $(f(2^{2^n}-1, i), f(0, i)) \in \mathbb{H}$ and $(f(i, 2^{2^n}-1), f(i, u)) \in \mathbb{V}$ for all $i < 2^{2^n}$.

We first implement the reduction using UCQs instead of CQs and then adapt the proof to CQs. In the previous reduction, the role name r was used to connect neighboring grid nodes and nodes in counting trees. In the current reduction, we replace r with the role composition $r^-; r$ where r^- denotes the inverse of r and which behaves like a reflexive-symmetric role. We use S as an abbreviation for $r^-; r$. In particular, $\exists S.C$ stands for $\exists r^-.\exists r.C$ and $\forall S.C$ stands for $\forall r^-.\forall r.C$. Some other details of the reduction are also different than before. Counting trees now have depth $m + 2$ instead of m, but no branching occurs on the last two levels of the tree. We also have three different versions of counting trees: one which uses the concept names $B_1, \overline{B}_1$ and $B_2, \overline{B}_2$ to store the two counters, one that uses $B_3, \overline{B}_3$ and $B_4, \overline{B}_4$, and one that uses $B_5, \overline{B}_5$ and $B_6, \overline{B}_6$. We say that the trees are of type 0, 1, or 2 to distinguish between the different versions. In the grid representation, we cycle through the types: from left to right and bottom to top, every tree of type 0 is succeeded by trees of type 1 which are succeeded by trees of type 2 which are succeeded by trees of type 3. Note that this refers to trees below grid nodes, but also to trees below horizontal and vertical step nodes. All this prepares for the construction of the UCQ later on.

Let P be a 2-exp torus tiling problem and w_0 an input to P of length n. We construct TBoxes $\mathcal{T}_1, \mathcal{T}_2$ and OMQs $Q_i = (\mathcal{T}_i, \mathbf{S}_E, q_i)$, $i \in \{1, 2\}$, such that $Q_1 \subseteq Q_2$ iff there is a tiling for P and w_0. The schema $\mathbf{S}_E$ consists of

- the EDB symbols r, B_i, $\overline{B}_i$, $i \in \{1, \ldots, 6\}$;
- concept names $A_0, \ldots, A_{m-1}$ and $\overline{A}_0, \ldots, \overline{A}_{m-1}$ to implement a binary counter that identifies the position of each leaf in a counting tree;
- concept names $L_0, \ldots, L_{m+2}$ that identify the levels in counting trees.

We now construct the TBox $\mathcal{T}_1$. We first define concept inclusions which verify that a grid node has a proper attached counting tree. We start with identifying nodes on level $m+2$ by the concept name $\mathsf{lev}_{m+2}^{G,t}$ where $t \in \{0, 1, 2\}$ describes the type of the counting tree as explained above. We only give the construction explicitly for $\mathsf{lev}^{G,0}_{m+2}$, which implements the two counters using $B_1, \overline{B}_1$ and $B_2, \overline{B}_2$:

$$A_i \sqsubseteq V_i \qquad \overline{A}_i \sqsubseteq V_i \qquad 0 \leq i < m$$
$$V_0 \sqcap \cdots \sqcap V_{m-1} \sqcap B_1 \sqcap B_2 \sqcap L_{m+2} \sqsubseteq \mathsf{lev}^{G,0}_{m+2}$$
$$V_0 \sqcap \cdots \sqcap V_{m-1} \sqcap \overline{B}_1 \sqcap \overline{B}_2 \sqcap L_{m+2} \sqsubseteq \mathsf{lev}^{G,0}_{m+2}.$$

To make the UCQ work later on, we need that level $m+1$-nodes are labeled complementarily regarding the concept names $A_i, \overline{A}_i, i \leq m$. We thus identify nodes on level $m+1$ as follows:

$$A_i \sqcap \exists S.(\mathsf{lev}^{G,0}_{m+2} \sqcap \overline{A}_i) \sqsubseteq A\mathsf{ok}_i \quad 0 \leq i \leq m$$
$$\overline{A}_i \sqcap \exists S.(\mathsf{lev}^{G,0}_{m+2} \sqcap A_i) \sqsubseteq A\mathsf{ok}_i \quad 0 \leq i \leq m$$
$$A\mathsf{ok}_0 \sqcap \cdots \sqcap A\mathsf{ok}_{m-1} \sqcap L_{m+1} \sqsubseteq \mathsf{lev}^{G,0}_{m+1}$$

Note that the first two lines may speak about different S-successors. It is thus not clear that they achieve the intended complementary labeling. Moreover, we have not yet made sure that level $m+2$-nodes are labeled with only one of $A_i, \overline{A}_i$ for each i and with only one $B_j, \overline{B}_j$ for each $j \in \{1,2\}$. We fix these problem by including the following concept inclusions in $\mathcal{T}_2$:

$$L_{m+1} \sqcap \exists S.(L_{m+2} \sqcap A_i) \sqcap \exists S.(L_{m+2} \sqcap \overline{A}_i) \sqsubseteq \bot$$
$$L_{m+1} \sqcap \exists S.(L_{m+2} \sqcap B_j) \sqcap \exists S.(L_{m+2} \sqcap \overline{B}_j) \sqsubseteq \bot$$

where i ranges over $0,\ldots,m-1$ and j over $\{1,2\}$. These inclusions ensure that all relevant successors are labeled identically regarding the relevant concept names: otherwise the instance is inconsistent with $\mathcal{T}_2$ and thus makes Q_2 true, which rules it out as a witness for non-containment.

We next make sure that every level m node has a level $m+1$ node attached and that its labeling is again complementary (in other words, the labeling of the level m node agrees with the labeling of the level $m+2$ node below the attached level $m+1$ node):

$$A_i \sqcap \exists S.(\mathsf{lev}^{G,0}_{m+1} \sqcap \overline{A}_i) \sqsubseteq A\mathsf{ok}'_i \quad 0 \leq i \leq m$$
$$\overline{A}_i \sqcap \exists S.(\mathsf{lev}^{G,0}_{m+1} \sqcap A_i) \sqsubseteq A\mathsf{ok}'_i \quad 0 \leq i \leq m$$
$$A\mathsf{ok}'_0 \sqcap \cdots \sqcap A\mathsf{ok}'_{m-1} \sqcap L_{m+1} \sqsubseteq \mathsf{lev}^{G,0}_{m}$$

We also include the following in $\mathcal{T}_2$:

$$L_m \sqcap \exists S.(L_{m+1} \sqcap A_i) \sqcap \exists S.(L_{m+1} \sqcap \overline{A}_i) \sqsubseteq \bot$$

where i ranges over $0,\ldots,m-1$. We next verify the remaining levels of the tree. To make sure that the required successors are present on all levels, we branch on the concept names A_i, $\overline{A}_i$ at level i of a counting tree and for all $j < i$, keep our choice of A_j, $\overline{A}_j$:

$$\exists S.(\mathsf{lev}^{G,0}_{i+1} \sqcap A_i) \sqcap \exists S.(\mathsf{lev}^{G,0}_{i+1} \sqcap \overline{A}_i) \sqsubseteq \mathsf{Succ}$$
$$A_j \sqcap \exists S.(\mathsf{lev}^{G,0}_{i+1} \sqcap A_j) \sqsubseteq \mathsf{Ok}_j$$
$$\overline{A}_j \sqcap \exists S.(\mathsf{lev}^{G,0}_{i+1} \sqcap \overline{A}_j) \sqsubseteq \mathsf{Ok}_i$$
$$\mathsf{Succ} \sqcap \mathsf{Ok}_0 \sqcap \cdots \sqcap \mathsf{Ok}_{i-1} \sqcap L_i \sqsubseteq \mathsf{lev}^{G,0}_{i}$$

where i ranges over $0,\ldots,m-1$ and j over $0,\ldots,i-1$. Again, lines one to three may speak about different successors and we need to make sure that all those successors are labeled identically. This is done by adding the following inclusions to $\mathcal{T}_2$:

$$L_i \sqcap \exists S.(L_{i+1} \sqcap A_j) \sqcap \exists S.(L_{i+1} \sqcap \overline{A}_j) \sqsubseteq \bot$$

where the ranges of i and j are as above. This finishes the verification of the counting tree. We do not use self step nodes in the current reduction, so a grid node is simply the root of a counting tree where both counter values are identical:

$$\mathsf{lev}^{G,0}_0 \sqsubseteq \mathsf{gactive}^0.$$

The superscript $\cdot^0$ in $\mathsf{gactive}^0$ indicates that the counting tree of which this node is the root is of type 0. Concept inclusions that set $\mathsf{gactive}^1$ and $\mathsf{gactive}^2$ are defined analogously, replacing $B_1, \overline{B}_1$ and $B_2, \overline{B}_2$ appropriately, as explained above. In a similar way, we can verify the existence of counting trees below horizontal step nodes and vertical step nodes, signalling the existence of such trees by the concept names $\mathsf{hactive}^t$ and $\mathsf{vactive}^t$, $t \in \{0,1,2\}$. In contrary to the counting trees between grid nodes, counting trees below horizontal and vertical step nodes need to properly increment the counters, as in the previous reduction. Details are slightly tedious but straightforward and thus omitted.

We also use $\mathcal{T}_1$ to enforce that all grid nodes are labeled with a tile type. However, as we shall see below we cannot use all nodes labeled with a $\mathsf{gactive}^t$-concept as grid nodes, but only those ones that have an S-neighbor which is labeled with $\mathsf{hactive}^{t\oplus 1}$ or with $\mathsf{vactive}^{t\oplus 1}$ where $\oplus$ denotes addition modulo three.[3] We call such nodes g-active and add the following to $\mathcal{T}_1$:

$$\mathsf{gactive}^t \sqcap \exists S.\mathsf{hactive}^{t\oplus 1} \sqsubseteq \bigsqcup_{T_i \in \mathbb{T}} T_i \qquad t \in \{0,1,2\}$$

We next add inclusions to $\mathcal{T}_1$ which identify a defect in the tiling and signal this by making the concept name D true:

1. horizontally neighboring tiles match; for all $T_i, T_j \in \mathbb{T}$ with $(T_i, T_j) \notin H$ and $t \in \{0,1,2\}$:

   $$T_i \sqcap \mathsf{gactive}^t \sqcap \exists S.(\mathsf{hactive}^{t\oplus 1} \sqcap \exists S.(\mathsf{gactive}^{t\oplus 2} \sqcap T_j)) \sqsubseteq D$$

2. vertically neighboring tiles match; for all $T_i, T_j \in \mathbb{T}$ with $(T_i, T_j) \notin V$ and $t \in \{0,1,2\}$:

   $$T_i \sqcap \mathsf{gactive}^t \sqcap \exists S.(\mathsf{vactive}^{t\oplus 1} \sqcap \exists S.(\mathsf{gactive}^{t\oplus 2} \sqcap T_j)) \sqsubseteq D$$

3. The tiling respects the initial condition. Let $w_0 = T_{i_0} \cdots T_{i_{n-1}}$. As in the previous reduction, it is tedious but not difficult to write concept inclusions to be included in $\mathcal{T}$ which ensure that, for $0 \leq i < n$, every element that is in $\mathsf{gactive}^t$ for some t and whose B_ℓ-value represents horizontal position i and vertical position 0, satisfies the concept name $\mathsf{pos}_{i,0}$. Here, (t,ℓ) ranges over $(0,1),(1,3),(2,5)$. We then add the following CQ to q_2 for $0 \leq j < n$ and all $T_\ell \in \mathbb{T}$ with $T_\ell \neq T_{i_j}$:

   $$\mathsf{pos}_{j,0} \sqcap T_\ell \sqsubseteq D.$$

[3]This is the reason why we reduce torus tiling instead of square tiling: in a square, grid nodes on the upper and right border are missing the required successors.

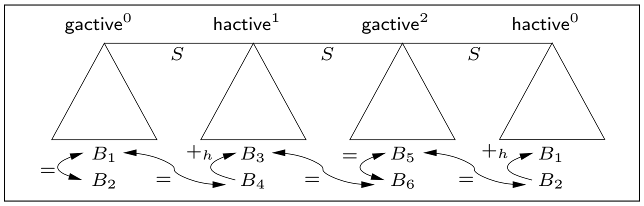


Figure 5: Counting strategy.

Note that Points 1 and 2 achieve the desired cycling through the three different types of counting trees: horizontal and vertical neighborships of g-active nodes whose types are not as expected are simply ignored (i.e., not treated as neighborships in the first place).

This completes the construction of the TBox $\mathcal{T}_1$. The query q_1 simply takes the form $\exists x\, D(x)$, thus Q_1 is true in an instance I iff the (potentially partial) grid in I does not admit a tiling, as desired. The construction of $\mathcal{T}_2$ is also finished at this point. It thus remains to construct q_2. As in the previous reduction, the purpose of q_2 is to ensure that counter values are copied appropriately to neighboring counting trees and that the two counter values below each grid and step node are unique. We call two counting trees *neighboring* if their roots are connected by the relation S. Since S is symmetric, we cannot distinguish successor counting trees from predecessor ones. The three different types of counting trees still allow us to achieve the desired copying of counter values. More precisely, we need to ensure that

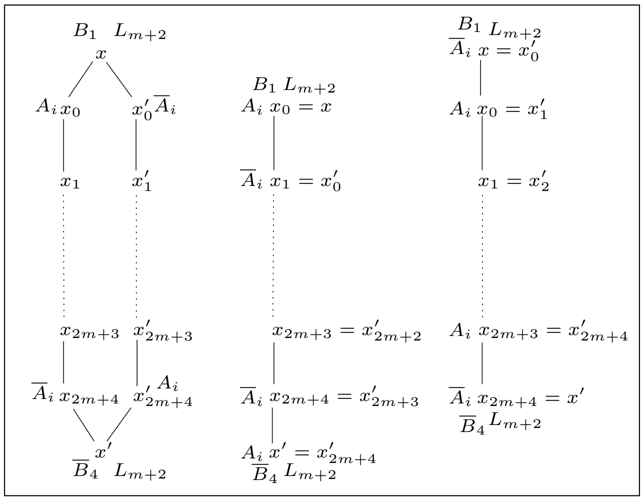
(Q1) the B_i-value of a counting tree coincides with the B_{i+3}-value of neighboring trees, for all $i \in \{1, 2, 3\}$;

(Q2) every g-active node is associated (via counting trees) with at most one B_i-value, for each $i \in \{1, \ldots, 6\}$.

The counting strategy is illustrated in Figure 5, displaying a horizontal fragment of the grid. Arrows annotated with "=" indicate identical counter values and arrows annotated with "$+_h$" indicate incrementation of the horizontal component of the counter. A vertical fragment would look identical, except that hactive^i is replaced with gactive^i and incrementation of the horizontal counter component with incrementation of the vertical component.

Before we start constructing q_2, we observe that it actually suffices to ensure (Q1) because (Q2) is then guaranteed automatically. The reason is that we are only interested in nodes that are g-active and thus have an S-neighbor which is labeled with hactive$^{t\oplus 1}$ or with vactive$^{t\oplus 1}$. Assume for example that a node a is in gactive^0 and has an S-neighbor b which is labeled with hactive^1. By (Q1), *all* leaves with a given position in the tree below a must agree regarding their $B_1, \overline{B}_1$-value with the $B_4, \overline{B}_4$-value of *all* leaves with the same position in the tree below b. Since both trees contain at least one leaf for each position, this achieves (Q2) for the B_1-value. It also achieves (Q2) for the B_2-value since that value is identical to B_1-value. Obviously, the other cases are analogous.

The UCQ q_1 for achieving (Q1) includes six CQs. We first construct a query q which is true in an instance I if



$B_1 \; L_{m+2}$
x
$A_i \, x_0$
$x_0' \, \overline{A}_i$
x_1
x_1'
x_{2m+3}
x_{2m+3}'
$\overline{A}_i \, x_{2m+4}$
$x_{2m+4}' \; A_i$
x'
$\overline{B}_4 \; L_{m+2}$
$B_1 \; L_{m+2}$
$A_i \; x_0 = x$
$\overline{A}_i \; x_1 = x_0'$
$x_{2m+3} = x_{2m+2}'$
$\overline{A}_i \; x_{2m+4} = x_{2m+3}'$
$A_i \; x' = x_{2m+4}'$
$\overline{B}_4 \; L_{m+2}$
$B_1 \; L_{m+2}$
$\overline{A}_i \; x = x_0'$
$A_i \; x_0 = x_1'$
$x_1 = x_2'$
$A_i \; x_{2m+3} = x_{2m+4}'$
$\overline{A}_i \; x_{2m+4} = x'$
$\overline{B}_4 \, L_{m+2}$

Figure 6: Component query for (Q1) and two identifications.

($*$) there are two leaves in neighboring counting trees that have the same position and such that one leaf is labeled with B_1 and the other one with $\overline{B}_4$.

The other five queries are then minor variations of q. We construct q from component queries $p_0, \ldots, p_{m-1}$, which all take the form of the query show on the left-hand side of Figure 6. Note that all edges are S-edges and that the only difference between the component queries is which concept names A_i and $\overline{A}_i$ are used. We assemble $p_0, \ldots, p_{m-1}$ into the desired query q by taking variable disjoint copies of $p_0, \ldots, p_{m-1}$ and then identifying (i) the x-variables of all components and (ii) the x'-variables of all components.

To see why q achieves ($*$), first note that the variables x and x' must be mapped to leaves of counting trees because of their L_{m+2}-label. Call these leaves a and a'. Since x is labeled with B_1 and x' with $\overline{B}_4$, a and a' must be in different trees. Since they are connected to x in the query, both x_0 and x_0' must then be mapped either to a or to its predecessor; likewise, x_{2m+4} and x_{2m+4}' must be mapped either to a' or to its predecessor. Because of the labeling of a and a' and the predecessors in the counting tree with A_i and $\overline{A}_i$, we are actually even more constrained: exactly one of x_0 and x_0' must be mapped to a, and exactly one of x_{2m+4} and x_{2m+4}' to a'. Since the paths between leaves in different trees in the instance have length at least $2m + 5$ and q contains paths from x_0 to x_{2m+4} and from x_0' to x_{2m+4}' of length $2m + 4$, only the following cases are possible:

- x_0 is mapped to a, x_0' to the predecessor of a, x_{2m+4}' to a', and x_{2m+4} to the predecessor of a';
- x_0' is mapped to a, x_0 to the predecessor of a, x_{2m+4} to a', and x_{2m+4}' to the predecessor of a'.

This gives rise to the two variable identifications in each query p_i shown in Figure 6. Note that the first case implies that a and a' are both labeled with A_i while they are both labeled with $\overline{A}_i$ in the second case. In summary, a and a' must thus agree on all concept names $A_i, \overline{A}_i$. Note that with the identification $x_0 = x$ (resp. $x_0' = x$), there is a path from x to x' in the query of length $2m + 5$. Thus, a and a' are

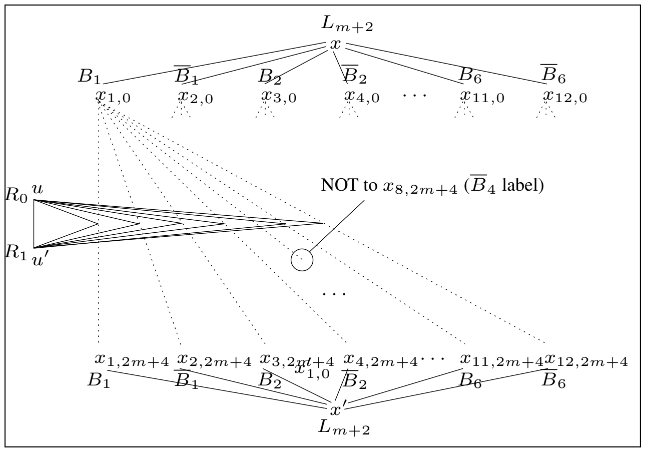


Figure 7: Additional component for CQ.

in neighboring counting trees. Since a must satisfy B_1 and a' must satisfy $\overline{B}_4$ due to the labeling of x and x', we have achieved $(*)$.

We now show how to replace the UCQ used in the reduction with a CQ. This requires the following changes:

1. every node on level m now has two successors instead of one (while nodes on level $m+1$ still have a single successor); in gactive^0-trees, one of the leafs below the same level m node carries the B_1,$\overline{B_1}$-label while the other leaf carries the B_2,$\overline{B_2}$-label; the labeling of the two leaves with A_i,$\overline{A}_i$ is identical; similarly for gactive^1 and gactive^2;
2. the predecessors of leaf nodes in counting trees receive additional labels: when the leaf node is labeled with B_i (resp. $\overline{B}_i$), then its predecessor is labeled with $\overline{B}_i$ (resp. B_i) and with B_j and $\overline{B}_j$ for all $j \in \{1, \dots, 6\} \setminus \{i\}$; these concept names are added to $\mathbf{S}_E$;
3. the roots of counting trees receive an additional label R_0 or R_1, alternating with neighboring trees; these concept names are added to $\mathbf{S}_E$, too;
4. the query construction is modified.

Points 2 and 3 are important for the CQ to be constructed to work correctly. Point 2 does not make sense without Point 1. Note that Points 1 to 3 can be achieved in a straightforward way by modifying the previous reduction, details are omitted. We thus concentrate on Point 4. The desired CQ q is again constructed from component queries. We use m components as shown in Figure 6, except that the B_1 and $\overline{B}_4$-labels are dropped. We additionally take the disjoint union with the component (partially) shown in Figure 7 where again x and x' are the variables shared with the other components. The dotted edges denote S-paths of length $2m+4$. For readability, we show only some of the paths. The general scheme is that every variable $x_{i,0}$ has a path to every variable $x_{j,2m+4}$ unless the two variables are labeled with complementary concept names, that is, with concept names B_i and $\overline{B}_j$ such that $i \in \{1,2,3\}$ and $j = i+1$ or with concept names $\overline{B}_i$ and B_j such that $i \in \{1,2,3\}$ and $j = i+1$. In the figure, we only show the paths outgoing from $x_{1,0}$. The edges that connect u and u' with the dotted paths always end at the middle point of a path, which has distance $m+2$ to the $x_{i,0}$ variable where the path starts and also distance $m+2$ to the $x_{j,2m+4}$ variable where it ends.

We have to argue that the CQ q just constructed achieves (Q1). As before, x and x' must be mapped to leafs of counting trees because of their L_{m+2}-label. Call these leaves a_1 and a_2. All $x_{i,0}$ must then be mapped to a or its predecessor, and all $x_{i,2m+4}$ must be mapped to a' or its predecessor. In fact, due to the labeling of a and a' and their predecessors in their counting tree with the concepts B_i, $\overline{B}_i$, exactly one variable $x_{i,0}$ from $x_{0,0}, \dots, x_{12,0}$ is mapped to a while all others are mapped to the predecessor of a; likewise, exactly one of the $x_{j,2m+4}$ from $x_{0,2m+4}, \dots, x_{12,2m+4}$ is mapped to a' while all others are mapped to the predecessor of a'. To achieve (Q1), we have to argue that $x_{i,0}$ and $x_{j,2m+4}$ are labeled with complementary concept names, and that a and a' are in neighboring computation trees.

Assume to the contrary that $x_{i,0}$ and $x_{j,2m+4}$ are not labeled with complementary concept names. Then they are connected in q by a path of length $2m+4$ whose middle point y is connected to the variables u and u'. In a homomorphism to the grid with counting trees, there are four possible targets for u and u' and for the predecessor y_{-1} of y on the connecting path and the successor y_1 of y on that path:

1. u, y_{-1} map to the same constant, and so do u' and y;
2. u, y map to the same constant, and so do u' and y_1;
3. u', y_{-1} map to the same constant, and so do u and y;
4. u', y map to the same constant, and so do u and y_1.

However, options 1 and 3 are impossible because there would have to be a path of length $2m+1$ from a node labeled R_0 or R_1 to the leaf a. Similarly, options 2 and 4 are impossible because there would have to be a path of length $2m+1$ from a node labeled R_0 or R_1 to the leaf a'. Thus, we have shown that $x_{i,0}$ and $x_{j,2m+4}$ are labeled with complementary concept names.

This together with the labeling scheme of Figure 5 also means that a and a' (to which $x_{i,0}$ and $x_{j,2m+4}$ are mapped) are not in the same counting tree. Moreover, they cannot be in counting trees that are further apart than one step because under the assumption that $x = x_{i,0}$ and $x' = x_{j,2m+4}$, there is a path of length $2m+5$ in the query from x to x'. Note that we can identify u with the $2m+2$nd variable on any such path and u' with the $2m+3$rd variable (or vice versa) to admit a match in neighboring counting trees.

An OMQ $Q = (\mathcal{T}, \mathbf{S}_E, q)$ is *FO-rewritable* iff there is an FO-query that is equivalent to Q. Rewritability into monadic Datalog and into unrestricted Datalog are defined accordingly.

Theorem 18. *In* $(\mathcal{ALC}, UCQ)$ *and* $(\mathcal{ALCI}, CQ)$*, rewritability into FO, into monadic Datalog, and into Datalog is* 2NExpTime*-hard.*

Proof. For $(\mathcal{ALC}, \text{UCQ})$, it suffices to note that every MDDLog program with only unary and binary EDB relations can be translated into an equivalent OMQ from

$(\mathcal{ALC}, \text{UCQ})$ in polynomial time (Bienvenu et al. 2014). Thus, the lower bounds for $(\mathcal{ALC}, UCQ)$ are an immediate consequence of Theorem 5.

For $(\mathcal{ALCI}, \text{CQ})$, we adapt the above hardness proof for containment, essentially in the same way as in the proof of Theorem 5. Our aim is thus to show that, from a 2-exp torus tiling problem P and an input w_0 to P, we can construct in polynomial time an $(\mathcal{ALCI}, \text{CQ})$-OMQ Q such that

1. if there is a tiling for P and w_0, then Q is FO-rewritable;
2. if there is no tiling for P and w_0, then Q is not Datalog-rewritable.

We have shown how to construct from a 2-exp torus tiling problem P and an input w_0 to P, two $(\mathcal{ALCI}, \text{CQ})$-OMQs Q_1, Q_2 such that $Q_1 \subseteq Q_2$ iff there is a tiling for P and w_0. Moreover, Q_2 consists only of inclusions of the form $C \sqsubseteq \bot$ with C an $\mathcal{ELI}$-concept (a concept built only from conjuntion and existential restrictions, possibly using inverse roles). As above, let $Q_i = (\mathcal{T}_i, \mathbf{S}_E, q_i)$ for $i \in \{1, 2\}$. The desired OMQ $Q = (\mathcal{T}, \mathbf{S}'_E, q)$ is constructed by choosing $q = q_2$, $\mathbf{S}'_E = \mathbf{S}_E \cup \{s, u\}$, and choosing for $\mathcal{T}$ the union of $\mathcal{T}_1$ and $\mathcal{T}_2$, extended with the following CIs:

1. $\exists u.D \sqsubseteq R \sqcup G \sqcup B$ (where D is the concept name used in q_1);
2. $C_1 \sqcap C_2 \sqsubseteq C_{q_2}$ for all distinct $C_1, C_2 \in \{R, G, B\}$ where C_{q_2} is an (easy to construct) $\mathcal{ELI}$-concept such that $C_{q_2}^{\mathcal{I}} \neq \emptyset$ implies $\mathcal{I} \models q_2$ for all interpretations $\mathcal{I}$;
3. $C \sqcap \exists s.C \sqsubseteq C_{q_2}$ for all $C \in \{R, G, B\}$.

We now show that Q satisfies Points 1 and 2 above.

For Point 1, assume that there is a tiling for P and w_0. Then $Q_1 \subseteq Q_2$. We claim that we obtain a UCQ-rewriting φ of Q by taking the disjunction of q_2 and of $\exists x\, C(x)$ for every CI $C \sqsubseteq \bot$ in $\mathcal{T}_2$. To see this, let I be an $\mathbf{S}'_E$-instance. Clearly, $I \models \varphi$ implies $I \models Q$. Conversely, assume that $I \models Q$. If $I \not\models Q_1$, then there is a model $\mathcal{I}$ of I and $\mathcal{T}_1$ such that $D^{\mathcal{I}} = \emptyset$. Thus the additional CIs from the construction of Q are inactive and $I \models Q$ implies $I \models Q_2$, thus $I \models \varphi$. Now assume $I \models Q_1$. Then $I \models Q_2$ since $Q_1 \subseteq Q_2$. Thus, again, $I \models \varphi$.

For Point 2, assume there is no tiling for P and w_0. Then $Q_1 \not\subseteq Q_2$. Given an undirected graph $G = (V, E)$, let the instance I_G^+ be defined as the disjoint union of the instance I_0 which represents the 2^{2^n}-grid plus counting gadgets, the instance I_G which contains a fact $s(v_1, v_2)$ for every $\{v_1, v_2\} \in V$, extended with the fact $u(v, g)$ for every $v \in V$ and element g of I_0.

Since there is no tiling for P and w_0, we have $I_0 \models Q_1$ and thus $I_G^+ \models Q_1$. By construction of Q and since $Q_1 \not\subseteq Q_2$, this implies that $I_G^+ \models Q$ iff G is not 3-colorable. It remains to argue that, consequently, a Datalog-rewriting of Q gives rise to a Datalog-rewriting of non-3-colorability (which doesn't exist). This can be done as in the proof of Theorem 5. ❑