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 €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 $\text{2NEXPTime}$-complete. In particular, this is the case for MDDLog, MMSNP, OMQs whose ontology is formulated in a DL between $\text{ALC}$ and $\text{SHI}$ and where the actual queries are UCQs, and OMQs whose ontology is formulated in a DL between $\text{ALCII}$ and $\text{SHI}$ 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 $\text{2ExpTime}$-complete (Benedikt, Bourhis, and Senellart 2012). Here, we show that adding disjunction increases the complexity to $\text{2NExpTime}$. 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 $\text{2NExpTime}$-hard and that containment in Boolean MDDLog is in $\text{2NExpTime}$. 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), and Schwentick 2008). For the upper bound, we assume that $\text{OMQ}$ containment is $\text{2NExpTime}$-hard. We refer to (Bourhis, Krötzsch, and Senellart 2012). Here, we show that adding disjunction increases the complexity to $\text{2NExpTime}$.

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 $\text{adom}(I)$ 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 $q(I) \subseteq \text{adom}(I)^n$, where $n \geq 0$ is the arity of $q$. If $n = 0$, then we say that $q$ is $\text{Boolean}$ and we write $I \models q$ if $() \in q(I)$.

We now introduce some concrete query languages. A conjunctive query (CQ) takes the form $\exists y \varphi(x, y)$ where $\varphi$ is a conjunction of relational atoms and $x, y$ denote tuples of variables. The variables in $x$ are called answer variables. Semantically, $\exists y \varphi(x, y)$ denotes the query

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

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 $\rho$ has the form

$S_1(x_1) \lor \cdots \lor S_n(x_n) \leftarrow R_1(y_1) \land \cdots \land R_n(y_n)$

where $n > 0$ and $m \geq 0$. We refer to $S_1(x_1) \lor \cdots \lor S_m(x_m)$ as the head of $\rho$, and to $R_1(y_1) \land \cdots \land R_n(y_n)$ as the body. Every variable that occurs in the head of a rule $\rho$ is required to also occur in the body of $\rho$. A disjunctive Datalog (DDLog) program $\Pi$ is a finite set of disjunctive Datalog rules with a selected goal relation $\text{goal}$ that does not occur in rule bodies and appears only in non-disjunctive goal rules $\text{goal}(x) \leftarrow R_1(x_1) \land \cdots \land R_n(x_n)$. The arity of $\Pi$ is the arity of the goal relation. Relation symbols that occur in the head of at least one rule of $\Pi$ are intensional (IDB) relations, and all remaining relation symbols in $\Pi$ are extensional (EDB) relations. Note that, by definition, goal is an IDB relation. A Datalog program is called monadic or an MDDLLog program if all its EDB relations except goal have arity at most one.

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

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

In some definitions of disjunctive datalog, empty rule heads (denoted $\bot$) are disallowed. We admit them only in our upper bound proofs, but do not use them for lower bounds, thus achieving maximum generality.
Let $\Pi_1, \Pi_2$ be DDLog programs over the same EDB schema $S_E$ and of the same arity. We say that $\Pi_1$ is contained in $\Pi_2$, written $\Pi_1 \subseteq \Pi_2$, if for every $S_E$-instance $I$, we have $\Pi_1(I) \subseteq \Pi_2(I)$.

**Example 1.** Consider the following DDLog program $\Pi_1$ over EDB schema $S_E = \{A, B, r\}$:

$$A_1(x) \lor A_2(x) \leftarrow A(x)$$
$$\text{goal}(x) \leftarrow A_1(x) \land r(x, y) \land A_1(y)$$
$$\text{goal}(x) \leftarrow A_2(x) \land r(x, y) \land A_2(y)$$

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

An MMSNP sentence over schema $S_E$ has the form $\exists x_1 \cdots \exists x_n \forall x_{n+1} \phi$ with $x_1, \ldots, x_n$ monadic second-order variables, $x_1, \ldots, x_m$ first-order variables, and $\phi$ a conjunction of formulas of the form

$$\alpha_1 \land \cdots \land \alpha_m \rightarrow \beta_1 \lor \cdots \lor \beta_m$$

where each $\alpha_i$ takes the form $X_i(x_j)$ or $R(x)$ with $R \in S_E$, and each $\beta_i$ takes the form $x_j(y)$. 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 $\phi$ can serve as a Boolean query in the obvious way, that is, $I \models \phi$ whenever $\phi$ 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 DDLog 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 $q_1 \leq q_2$ if and only if $\neg q_1 \not\leq \neg q_2$. Consequently, any upper bound for containment in DDLog also applies to MMSNP and so does any lower bound for containment between Boolean DDLog programs.

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

$$\top \rightarrow R(x) \lor G(x) \lor B(x)$$
$$C(x) \land r(x, y) \land C(y) \rightarrow \bot$$

is equivalent to the Boolean DDLog program

$$r(x, y) \rightarrow C(x) \lor \neg C(x) \lor B(x) \quad \text{for } C \in \{R, G, B\}$$
$$r(x, y) \rightarrow C(x) \lor \neg C(y) \lor B(x) \quad \text{for } C \in \{R, G, B\}$$
$$\neg R(x) \land \neg C(x) \land \neg B(x) \rightarrow \text{goal}()$$
$$C(x) \land r(x, y) \land C(y) \rightarrow \text{goal}() \quad \text{for } C \in \{R, G, B\}.$$

**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 $P = (\mathbb{T}, \Pi, \mathbb{V})$ where

- $\mathbb{T} = \{T_1, \ldots, T_p\}, p \geq 1$, 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 $w = T_{i_1} \cdots T_{i_n}$. A tiling for $P$ and $w$ is a map $f : \{0, \ldots, 2^{2^n} - 1\} \times \{0, \ldots, 2^{2^n} - 1\} \rightarrow \mathbb{T}$ such that $f(0,0) = T_{i_j}$ 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 $P$ and a Boolean CQ $q$ such that $\Pi \not\subseteq q$ if 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. $P$ 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 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
navigation gadget $r(a, a_1), r(a_1, a_2), \text{jump}(a, a_2)$ and every right successor $a$ has an attached right navigation gadget $r(a, a_1), \text{jump}(a, a_1)$—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 $\Pi$. As the EDB schema, we use $S_E = \{r, \text{jump}, B_1, B_2, \overline{B}_1, \overline{B}_2\}$ where $r$ and $\text{jump}$ 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):

\[
\begin{align*}
\text{left}(x) & \leftarrow r(x, y) \land r(y, z) \land \text{jump}(x, z) \\
\text{right}(x) & \leftarrow r(x, y) \land \text{jump}(x, y) \\
\text{lrok}(x) & \leftarrow \text{left}(x) \\
\text{lrok}(x) & \leftarrow \text{right}(x) \\
\text{lev}_{m}^{G}(x) & \leftarrow B_{1}(x) \land B_{2}(x) \land \text{lrok}(x) \\
\text{lev}_{m}^{G}(x) & \leftarrow \overline{B}_{1}(x) \land \overline{B}_{2}(x) \land \text{lrok}(x) \\
\text{lev}_{i+1}^{G}(y) & \leftarrow r(x, y_1) \land \text{lev}_{i+1}^{G}(y_1) \land \text{left}(y_1) \land \text{right}(y_2) \\
\text{lev}_{i+1}^{G}(y) & \leftarrow r(x, y_2) \land \text{lev}_{i+1}^{G}(y_2) \\
\end{align*}
\]

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

\[
\text{gactive}(x) \leftarrow \text{lev}_{m}^{G}(x) \land r(x, y) \land \text{lev}_{0}^{G}(y) \land r(y, x)
\]

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 $\text{lev}_{i}^{G}$ 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 $\text{lev}_{i}^{G}$, we use relations $\text{lev}_{i}^{H, \ell}$ where $\ell \in \{1, 2, 3\}$ indicates that the leaf word of the subtree belongs to the language $L_\ell$:

\[
\begin{align*}
\text{lev}_{m}^{H, 1}(x) & \leftarrow \overline{B}_{1}(x) \land B_{2}(x) \land \text{lrok}(x) \\
\text{lev}_{m}^{H, 2}(x) & \leftarrow B_{1}(x) \land \overline{B}_{2}(x) \land \text{lrok}(x) \\
\text{lev}_{m}^{H, 3}(x) & \leftarrow B_{1}(x) \land B_{2}(x) \land \text{lrok}(x) \\
\text{lev}_{i+1}^{H, \ell}(y) & \leftarrow r(x, y_1) \land \text{lev}_{i+1}^{H, \ell}(y_1) \land \text{left}(y_1) \land \text{right}(y_2) \\
\text{lev}_{i+1}^{H, \ell}(y) & \leftarrow r(x, y_2) \land \text{lev}_{i+1}^{H, \ell}(y_2) \\
\end{align*}
\]

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:

\[
\text{hactive}(x) \leftarrow r(x, y_1) \land \text{lev}_{m}^{H, \ell_1}(y_1) \land \text{left}(y_1) \land \text{right}(y_2)
\]

We omit the rules for the corresponding IDB relation vactive. Call the fragment of $\Pi$ that we have constructed up to this point $\Pi_{\text{tree}}$. Before proceeding with the definition of $\Pi$, we need to define the CQ $q$ because it is used as a rule body in $\Pi$. Recall
that we want $q$ 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 construct a UCQ and show in the appendix how to replace it with a CQ, which also involves some minor additions to the program $\Pi_{\text{tree}}$ 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 $0 < i \leq m$,

\[
q_i(x_i, y_i) = \exists y_{i-1} \exists y_{i-2} \exists z_{i,0} \cdots \exists z_{i,i+2} \exists z'_{i,1}, \cdots \exists z'_{i,i+3} \\
q_{i-1}(x_{i-1}, y_{i-1}) \land r(x_{i-1}, x_i) \land r(y_{i-1}, y_i) \land \\
\text{jump}(x_i, z_{i,i+2}) \land \text{jump}(y_i, z'_{i,i+3}) \land \\
r(z_{i,0}, z_{i,1}) \land \cdots r(z_{i,i+1}, z_{i,i+2}) \land \\
r(z_{i,0}, z'_{i,1}) \land r(z_{i,1}, z'_{i,2}) \land \cdots r(z'_{i,i+2}, z'_{i,i+3})
\]

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):

\[
q = \exists x_m \exists y_m q_m(x_m, y_m) \land B_1(x_m) \land B_2(y_m) \\
\lor \exists x_m \exists y_m q_m(x_m, y_m) \land \overline{B}_1(x_m) \land B_2(y_m)
\]

The first CQ in $q$ is displayed in Figure 2.

We now proceed with the construction of $\Pi$. Recall that we want an instance to make $\Pi$ true if it admits no tiling for $P$ and $w$. We thus label all g-active nodes with a tile type:

\[
\bigvee_{T_i \in \mathbb{T}} T_i(x) \leftarrow \text{gactive}(x)
\]

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 $(T_i, T_j) \notin \mathcal{H}$:

\[
\text{goal()} \leftarrow T_i(x) \land \text{gactive}(x) \land r(x, y) \land \text{hactive}(y) \land \\
r(y, z) \land T_j(z) \land \text{gactive}(z)
\]

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

\[
\text{goal()} \leftarrow T_i(x) \land \text{gactive}(x) \land r(x, y) \land \text{vactive}(y) \land \\
r(y, z) \land T_j(z) \land \text{gactive}(z)
\]

Figure 2: The first CQ in $q$.

The last kind of defect concerns the initial condition. Let $w_0 = T_{00} \cdots T_{n-1}$. It is tedious but not difficult to write rules which ensure that, for all $i < n$, every $g$-active element whose $B_i$-value represents horizontal position $i$ and vertical position 0 satisfies the monadic IDB relation $\text{pos}_{0,0}$. We then put for all $j < n$ and all $T_i \in \mathbb{T}$ with $T_i \neq T_j$:

\[
\text{goal()} \leftarrow \text{pos}_{0,0}(x) \land T_i(x).
\]

This completes the construction.

**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 \textit{FO-rewritable} if there is an equivalent first-order query and (monadic) \textit{Datalog-rewritable} if there is an equivalent (non-disjunctive) (monadic) Datalog query. \textit{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 CSMPs 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 CSMPs 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, $\Pi_1$ and $\Pi_2$ are converted into a simplified form, then containment between the resulting programs $\Pi_1^S$ and $\Pi_2^S$ 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 $\Pi_1^S$ and $\Pi_2^S$ 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 with girth in all three steps and can finally resolve the problem in the last step.

We start with introducing the notion of girth. Let $I$ be an instance. A subset $C \subseteq I$ is a cycle if either it consists of a single fact $R_i(a_1, \ldots, a_n)$ with $a_i = a_j$ for some distinct $i, j$ or it contains at least two facts and the facts in $C$ can be ordered into $R_1(a_1), \ldots, R_n(a_n)$ such that there exist distinct constants $a_1, \ldots, a_n$ with $a_i \cap a_{i+1} \neq \emptyset$ for $1 \leq i \leq n$ and $a_n+1 := a_1$. The length of a cycle is one for cycles of the former kind and $n$ for cycles of the latter kind. The girth of $I$ is the length of the shortest cycle contained in it and $\infty$ if $I$ contains no cycle. For MDDLog programs $\Pi_1, \Pi_2$ over the same EDB schema and $k \geq 0$, we write $\Pi_1 \subseteq_{k} \Pi_2$ if $\Pi_1(I) \subseteq \Pi_2(I)$ for all $S$-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 $\Pi$, denoted $|\Pi|$, is the number of symbols needed to write $\Pi$ 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 $\Pi$. The atom width (resp. variable width) of $\Pi$ is the maximum number of atoms in any rule body (resp. variables in any rule) in $\Pi$.

### From Unrestricted to Simple Programs

An MDDLog program $\Pi^S$ is simple if it satisfies the following conditions:

1. every rule in $\Pi^S$ 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 $\Pi_1, \Pi_2$ be Boolean MDDLog programs over EDB schema $S_E$. Then one can construct simple Boolean MDDLog programs $\Pi_1^S, \Pi_2^S$ over EDB schema $S'_E$ such that

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

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 $s$ is the rule size, then

3. $|\Pi_1^S| \leq p(r \cdot 2^s)$;
4. the variable width of $\Pi_1^S$ is bounded by that of $\Pi_1$;
5. $|S_E'| \leq p(r \cdot 2^s)$;

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

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 $\Pi$. In the first step, we extend $\Pi$ with all rules that can be obtained from a rule in $\Pi$ by consistently identifying variables. We then split up each rule in $\Pi$ 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 $R(x_1, \ldots, x_n)$ 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:

$$P(x_3) \leftarrow A(x_1) \land r(x_1, x_2) \land r(x_2, x_3) \land r(x_3, x_1)$$

$$\text{goal}() \leftarrow r(x_1, x_2) \land r(x_2, x_3) \land r(x_3, x_1) \land P(x_1) \land P(x_2) \land P(x_3)$$

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

$$P(x_3) \leftarrow R_{q_1}(x_1, x_2, x_3) \land R_{q_2}(x_1, x_2, x_3) \land P(x_1) \land P(x_2) \land P(x_3)$$

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 $S_{E'}$-instance $I$ defined by

$$R_{q_1}(a, a', c'), R_{q_2}(b, b', a'), R_{q_3}(c, c', b').$$

The goal rules from the simplified program do not apply. But when translating into an $S_{E'}$-instance $I$ 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 $R_{q_2}(a', b', c'), R_{q_3}(b', c', a')$, $R_{q_3}(c', a', b')$ that are ‘missed’ in $I$. Such effects can only happen on instances whose girth is at most $w$, such as $I$. 

212
From Containment to Relativized Emptiness  A disjointness constraint is a rule of the form $\bot \leftarrow P_1(x) \land \cdots \land P_n(x)$ where all relations are of the same arity and at most unary. Let $\Pi$ be a Boolean MDDLog program over EDB schema $S_E$ and $D$ a set of disjointness constraints over $S_E$. We say that $\Pi$ is semi-simple w.r.t. $D$ if $\Pi$ is simple when all relations that occur in $D$ are viewed as IDB relations. We say that $\Pi$ is empty w.r.t. $D$ if for all $S_E$-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 $\Pi$ and a set of disjointness constraints $D$ such that $\Pi$ is semi-simple w.r.t. $D$, whether $\Pi$ is empty w.r.t. $D$.

**Theorem 7.** Let $\Pi_1, \Pi_2$ be simple Boolean MDDLog programs over EDB schema $S_E$. Then one can construct a Boolean MDDLog program $\Pi$ over EDB schema $S_E'$ and a set of disjointness constraints $D$ over $S_E'$ such that $\Pi$ is semi-simple w.r.t. $D$ and

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

Moreover,

3. $|\Pi| \leq |\Pi_1| \cdot 2^{|S_{E,2}| - 1}, |D| \leq O(|S_{1,2}|)$;
4. the variable width of $\Pi \cup D$ is bounded by the variable width of $\Pi_1 \cup \Pi_2$;
5. $|S_E'| \leq |S_E| + |\Pi_2|$.

where $v_1$ is the variable width of $\Pi_1$ and $S_{1,2}$ is the IDB schema of $\Pi_2$. 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 $\Pi_1, \Pi_2$ be simple Boolean MDDLog programs over EDB schema $S_E$. For $i \in \{1, 2\}$, let $S_{i,1}$ be the set of IDB relations in $\Pi_i$ with goal relations $goal_i \in S_{i,1}$ and assume w.l.o.g. that $S_{1,1} \cap S_{2,1} = \emptyset$. Set $S_E' := S_E \cup S_{1,2} \cup \{P \mid P \in S_{1,2}\}$. The MDDLog program $\Pi$ is constructed in two steps. We first add to $\Pi$ every rule that can be obtained from a rule $\rho$ in $\Pi_1$ by extending the rule body with

- $P(x)$ or $\overline{P}(x)$, for every variable $x$ in $\rho$ and every unary $P \in S_{1,2}$, and
- $P()$ or $\overline{P}()$ for every nullary $P \in S_{1,2}$; for $P = goal_2$, we always include $\overline{P}()$ but never $P()$.

In the second step of the construction, we remove from $\Pi$ every rule $\rho$ whose body being true implies that a rule from $\Pi_2$ is violated, that is, there is a rule whose body is the CQ $q(x)$ and with head $P_1(y_1) \lor \cdots \lor P_n(y_n)$ and a variable substitution $\sigma$ such that

- $\sigma(q)$ is a subset of the body of $\rho$ and
- $\overline{P}(\sigma(y_i))$ is in the body of $\rho$, for $1 \leq i \leq n$.

The goal relation of $\Pi$ is $goal_1()$. The set of disjointness constraints $D$ then consists of all rules $\bot \leftarrow P(x) \lor \overline{P}(x)$ for each unary $P \in S_{1,2}$ and $\bot \leftarrow P() \land \overline{P}()$ for each nullary $P \in S_{1,2}$. It is not hard to verify that $\Pi$ and $D$ satisfy the size bounds from Theorem 7. We show in the appendix that $\Pi$ satisfies Points 1 and 2 of Theorem 7.

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

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

Let $\Pi$ be a Boolean MDDLog program over EDB schema $S_E$ and let $D$ be a set of disjointness constraints over $S_E$ such that $\Pi$ is semi-simple w.r.t. $D$. To prove Theorem 8, we show that we can construct a finite set of $S_E$-instances satisfying $D$ such that $\Pi$ is empty w.r.t. $D$ if and only if it is empty in the constructed set of instances. Let $S_D$ be the set of all EDB relations that occur in $D$. For $i \in \{0, 1\}$, an $i$-type is a set of $i$-ary relation symbols from $S_D$ 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 $\theta$ of all nullary $P \in S_D$ 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 $P \in S_D$ with $P(a) \in I$.

We build an $S_E$-instance $\Pi_0$ for each 0-type $\theta$. The elements of $\Pi_0$ are exactly the 1-types and $K_\theta$ consists of the following facts:

- $P(t)$ for every 1-type $t$ and each $P \in t$;
- $R(t_1, \ldots, t_n)$ for every relation $R \in S_E \setminus S_D$ and all 1-types $t_1, \ldots, t_n$;
- $P()$ for each nullary $P \in \theta$.

Note that, by construction, $\Pi_0$ is an $S_E$-instance that satisfies all constraints in $D$.

**Lemma 9.** $\Pi$ is empty w.r.t. $D$ iff $K_{\Pi_0} \not\models \Pi$ for all 0-types $\theta$.

By Lemma 9, we can decide emptiness of $\Pi$ by constructing all instances $K_\theta$ of $K_\theta$ to the IDB relations in $\Pi$ that does not contain the goal relation, and then verifying by an iteration over all possible homomorphisms from rule bodies in $\Pi$ to $K_\theta$ that all rules in $\Pi$ are satisfied in $K_\theta$.

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

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 $\Pi$ over EDB schema $S_E$ and set of disjointness constraints $D$ over $S_E$ such that $\Pi$ is semi-simple w.r.t. $D$, the following are equivalent for any $g \geq 0$:

1. $\Pi$ is empty regarding $D$ and
2. $\Pi$ is empty regarding $D$ and instances of girth exceeding $g$.

The proof of Lemma 11 uses a translation of semi-simple MDDLog programs with disjonctivity 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 $\Pi_2$ to the complexity is exponentially larger than that of $\Pi_1$.

**Theorem 12.** Containment between Boolean MDDLog programs and between MMSNP sentences is in 2NExpTime. More precisely, for Boolean MDDLog programs $\Pi_1$ and $\Pi_2$, it can be decided non-deterministically in time $2^{2^{2\left(|\Pi_1| \log |\Pi_1|\right)}}$ 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 ally generalize Theorem 12 by allowing both constants in

1. replacing every occurrence of a term in the body of a rule with a fresh variable and every occurrence of a term $t$ in the head of a rule with one of the variables introduced for $t$ in the rule body;

2. adding $R(x)$ to the rule body whenever some occurrence of a variable $x_0$ in the original rule has been replaced with $x$ and $\delta(x_0) = R$.

For example, the rule $P_1(y) \lor P_2(y) \leftarrow r(x, y, y) \land s(y, z)$ in $\Pi_1$ gives rise, among others, to the following rule in $\Pi_2$:

$P_1(y) \lor P_2(y) \leftarrow r(x_1, y_1, y_2) \land s(y, z_1) \land R_{a_1}(x_1) \land R_{a_2}(y_1) \land R_{a_2}(y_2) \land R_{a_2}(y_3) \land R_{\delta(z_1)}(z_1)$.  

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 $C$. Note that the original variable $y$ has been duplicated because $R_{a_2}$ needs not be a singleton set while $a_2$ denotes a single object. So intuitively, $\Pi_2$ treats its input instance $I$ as if it was the quotient $I'/I$ of $I$ obtained by identifying all $a_1$, $a_2$ with $R_0(a_1), R_0(a_2)$ in $I$ for some $b \in C$. In addition to the above rules, $\Pi_2'$ also contains

$\text{goal}() \leftarrow R_{a_1}(x) \land R_{a_2}(x)$ and $\text{goal}() \leftarrow R_{a_1}(x) \land R_{\delta(x)}(x)$ for all distinct $a_1, a_2 \in C$.

**Lemma 14.** $\Pi_1 \subseteq \Pi_2$ iff $\Pi_1' \subseteq \Pi_2'$.

It can be verified that $|\Pi_1'| \leq 2^{2|\Pi_1|^2}$ and that the rule size of $\Pi_1'$ is bounded by twice the rule size of $\Pi_1$. Because of the latter, the simplification of the programs $\Pi_1'$ according to Theorem 6 yields programs whose size is still bounded by $2^{2(|\Pi_1|)}$, as in the proof of Theorem 12, and whose variable width is bounded by twice the variable width of $\Pi_1$. 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 $\Pi_1$ and $\Pi_2$, it can be decided non-deterministically in time $2^{2^{2(|\Pi_2| \log |\Pi_1|)}}$ 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 $S_E$ with is a triple $(T, S_E, q)$, where $T$ is a TBox formulated in a description logic and $q$ is a query over the schema $S_E \cup \text{sig}(T)$, with $\text{sig}(T)$ the set of relation symbols used in $T$. The TBox can introduce symbols that are not in $S_E$, which allows it to enrich the schema of the query $q$. As the TBox language, we use the description logic $\mathcal{ALCI}$, 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 \((\mathcal{ALC}, \text{UCQ})\), \((\mathcal{ALCI}, \text{UCQ})\), \((\text{SHI}, \text{UCQ})\), and so on. In OMQs \((T, S_E, q)\) from \((\text{SHI}, \text{UCQ})\), 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 \(Q = (T, S_E, q)\) be given by

\[
T = \{ \exists \text{manages}. \text{Project} \sqsubseteq \text{Manager}, \text{Employee} \sqsubseteq \text{Male} \sqcup \text{Female} \}
\]

\[
S_E = \{ \text{Employee}, \text{Project}, \text{Male}, \text{Female}, \text{manages} \}
\]

\[
q(x) = \text{Manager}(x) \land \text{Female}(x)
\]

On the \(S_E\)-instance

\[
\text{manages}(e_1, e_2), \text{Female}(e_1), \text{Project}(e_2),
\]

\[
\text{manages}(e_1', e_2), \text{Employee}(e_1'),
\]

the only certain answer to \(Q\) is \(e_1\).

Let \(Q_i = (T_i, S_{E_i}, q_i), i \in \{1, 2\}\). Then \(Q_1\) is contained in \(Q_2\), written \(Q_1 \subseteq Q_2\), if for every \(S_{E_i}\)-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 \(S_{E_i}\)-instances, but only to those that are consistent with both \(T_1\) and \(T_2\). 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 \(|T|\) and \(|q|\) to denote the size of a TBox \(T\) 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 \(2\text{NExpTime}-\)complete:

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

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

We start with the lower bounds. For Point 1 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 \((\text{SHI}, \text{UCQ})\) 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 bound stated in Point 1 of Theorem 17 is a consequence of those in Theorem 3. It can be shown exactly as in the proof of Theorem 3 in (Bienvenu, Lutz, and Wolter 2012) that we can assume the TBoxes of the two OMQs to be identical.

Now for the lower bound stated in Point 2 of Theorem 17. The proof is 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 given in the appendix.

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

**Theorem 18.** In \((\mathcal{ALC}, \text{UCQ})\) and \((\mathcal{ALCI}, \text{CQ})\), rewritability into \(\text{FO}\), into monadic Datalog, and into Datalog is \(2\text{NExpTime}\)-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 \(Q = (T, S_E, q)\) from \((\text{SHI}, \text{UCQ})\), one can construct an equivalent MDDLog program \(\Pi\) such that

1. \(|\Pi| \leq 2^{p(|q| \log |T|)}\);
2. the IDB schema of \(\Pi\) is of size \(2^n(|q| \log |T|)\);
3. the rule size of \(\Pi\) 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 \((\text{SHI}, \text{UCQ})\). 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 \(2\text{NExpTime}\) upper bound. In contrast to Theorem 12, though, we only get an algorithm whose running time is double exponential in both inputs \((T_1, S_{E_1}, q_1)\) and \((T_2, S_{E_2}, q_2)\). 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 \(T_1\) and \(T_2\). This is good news since the size of \(q_1\) and \(q_2\) is typically very small compared to the sizes of \(T_1\) and \(T_2\). 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 \(\text{NExpTime}\) upper bound for containment.

**Theorem 20.** Containment between OMQs from \((\text{SHI}, \text{UCQ})\) is in \(2\text{NExpTime}\). More precisely, for
OMQs $Q_1 = (T_1, S_E, q_1)$ and $Q_2 = (T_2, S_E, q_2)$, it can be decided non-deterministically in time $2^{2^{pq(\|q_1\|,\|q_2\|,\|T_1\|,\|T_2\|)}}$ 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, see (Baget et al. 2009)) seems not too hard.

Other natural open problems concern containment between OMQs. In particular, it would be good to know the complexity of containment in $(\text{ALC}, \text{CQ})$ which must lie between $\text{NEXPTime}$ and $\text{2NEXPTime}$. Note that our first lower bound crucially relies on unions of conjunctive queries to be available, and the second one on inverse roles. It is known that adding inverse roles to $\text{ALC}$ tends to increase the complexity of querying-related problems (Lutz 2008), so the complexity of containment in $(\text{ALC}, \text{CQ})$ might indeed be lower than $\text{2NEXPTime}$. It would also be interesting to study containment for OMQs from $(\text{ALCIT}, \text{UCQ})$ 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.** Pierre Bourhis was supported by CPER Data and Carsten Lutz was supported by ERC grant 647289 CODA.

**References**


