ABSTRACT

While checking containment of Datalog programs is undecidable, checking whether a Datalog program is contained in a union of conjunctive queries (UCQ), in the context of relational databases, or a union of conjunctive 2-way regular path queries (UC2RPQ), in the context of graph databases, is decidable. The complexity of these problems is, however, prohibitive: \(2\text{EXPTIME}\)-complete. We investigate to which extent restrictions on UCQs and UC2RPQs, which have been known to reduce the complexity of query containment for these classes, yield a more “manageable” single-exponential time bound, which is the norm for several static analysis and verification tasks.

Checking containment of a UCQ \(\Theta'\) in a UCQ \(\Theta\) is NP-hard, in general, but better bounds can be obtained if \(\Theta\) is restricted to belong to a tractable class of UCQs, e.g., a class of bounded treewidth or hypertreewidth. Also, each Datalog program \(\Pi\) is equivalent to an infinite union of UCQs. This motivated us to study the question of whether restricting \(\Theta\) to belong to a tractable class also helps alleviate the complexity of checking whether \(\Pi\) is contained in \(\Theta\).

We study such question in detail and show that the situation is much more delicate than expected: First, tractability of UCQs does not help in general, but further restricting \(\Theta\) to be acyclic and have a bounded number of shared variables between atoms yields better complexity bounds. As corollaries, we obtain that checking containment of \(\Pi\) in \(\Theta\) is in \text{EXPTIME} if \(\Theta\) is of treewidth one, or it is acyclic and the arity of the schema is fixed. In the case of UC2RPQs we show an \text{EXPTIME} bound when queries are acyclic and have a bounded number of edges connecting pairs of variables. As a corollary, we obtain that checking whether \(\Pi\) is contained in UC2RPQ \(\Gamma\) is in \text{EXPTIME} if \(\Gamma\) is a strongly acyclic UC2RPQ. Our positive results for UCQs and UC2RPQs are optimal, in a sense, since slightly extending the conditions turns the problem \(2\text{EXPTIME}\)-complete.

1. INTRODUCTION

Query containment is a basic static analysis task that amounts to check whether the evaluation of a query \(q\) is necessarily contained in the evaluation of another query \(q'\) (often written as \(q \subseteq q'\)). Several database tasks crucially depend on the ability to check query containment; these include, e.g., query optimization, view-based query answering, querying incomplete databases, integrity checking, and implication of dependencies: cf. [6, 11, 20, 22, 23, 27].

Checking containment between queries is difficult computationally. For instance, the containment problem is undecidable for queries expressible in first-order logic (FO), and, thus, for any relational language that contains FO (such as SQL). Decidability results can be obtained for syntactically restricted classes of FO formulas. In the context of databases, the most important such restriction is defined by the class of conjunctive queries (CQs), or, equivalently, the expressions defined by the select-project-join operators of relational algebra. It follows from the seminal work of Chandra and Merlin [10] that CQ containment and CQ evaluation are polynomially equivalent problems and both are NP-complete. The NP upper bound is not affected if we extend the language to also consider unions of CQs (UCQs).

Extending the class of UCQs with recursion yields the important language of Datalog, which has gained renewed attention in the last years due to its applications in areas such as distributed computing, ontological reasoning, cluster computing, graph databases, and data exchange (cf. [19, 2] for a broad description of current applications of Datalog in academia and industry). This extension, however, has a crucial drawback: the containment problem for Datalog is undecidable [33]. This has motivated the search for relevant restrictions of the problem that can be effectively decided, ideally at a reasonable computational cost.

Since containment for UCQs is decidable, a natural restriction of the Datalog containment problem is when one of the two queries is a UCQ. This actually yields positive results in any of the two possible cases: Checking whether a UCQ \(\Theta\) is contained in a Datalog program \(\Pi\) is \(2\text{EXPTIME}\)-complete [16], while the opposite, deciding whether the Datalog program \(\Pi\) is contained in the UCQ \(\Theta\), is \(\text{EXPTIME}\)-complete [12]. From this we obtain that the important problem of checking whether the recursive program \(\Pi\) is equivalent to the nonrecursive (and, thus, typically easier to evaluate and optimize) query \(\Theta\) can be solved in \(2\text{EXPTIME}\).

While the decidability of containment of Datalog in UCQs is of theoretical importance, it is impractical due to the unavoidable double exponential cost. In addition, the \(2\text{EXPTIME}\) lower bound
holds even for UCQs of fixed arity, and, therefore, it implies “real” intractability. This raised the interest in identifying which parameters of the input affect its complexity. In particular, Chaudhuri and Vardi [13] and Benedikt et al. [5] performed a fine analysis of the complexity of the problem for restricted classes of Datalog programs (e.g. monadic, linear, single-rule and non-persistent programs), identifying cases that lead to better bounds.

In this paper we study the opposite question: To what extent is it possible to obtain more practical bounds for the problem by restricting the syntactic shape of UCQs, while retaining the full expressive power of Datalog programs? Our goal is identifying relevant restrictions on UCQs for which the double-exponential time procedure from [12] can be replaced by a more “acceptable” single-exponential time one, which is the norm in many static analysis and verification questions [28, 30].

Consider first the problem of checking whether \( \theta \subseteq \theta' \) for CQs \( \theta \) and \( \theta' \). This problem is NP-complete, but it follows from [10] that it can be solved efficiently if \( \theta' \) belongs to a tractable class of CQs; that is, a class of CQs whose evaluation problem can be solved in polynomial time. Due to a flurry of activity in the last two decades, we have now a fairly complete picture of what these classes are:

1. Chekuri and Rajaraman [14] (see also [18]) proved that each level of the hierarchy
   \[
   TW(1) \subset TW(2) \subset \cdots \subset TW(k) \subset \cdots,
   \]
   where \( TW(k) \) is the set of CQs of treewidth at most \( k \), is tractable. Treewidth is a well-studied notion that provides a measure for the tree-likeness of a CQ.

2. The treewidth of a CQ is defined in terms of its underlying graph, but this is too restrictive when the schema is not fixed. In such case, it is more convenient to measure the tree-likeness of a CQ in terms of its underlying hypergraph. Gottlob et al. [24] identified an appropriate correlate of treewidth in this context — called hypertreewidth — and proved that each level of the hierarchy
   \[
   HW(1) \subset HW(2) \subset \cdots \subset HW(k) \subset \cdots,
   \]
   where \( HW(k) \) corresponds to the class of CQs of hypertreewidth at most \( k \), is tractable. Interestingly, the first level of this hierarchy, \( HW(1) \), corresponds to the well-known class \( AC \) of acyclic CQs [36]. (See [25] for some extensions of this line of work.)

We can easily extend these two notions to UCQs: A UCQ \( \Theta \) is in \( TW(k) \) (resp. \( HW(k) \)) iff each one of the CQs in \( \Theta \) is in \( TW(k) \) (resp. \( HW(k) \)). Tractability results for query evaluation and containment extend to these classes [31].

Returning to the problem of checking containment of a Datalog program \( \Pi \) in a UCQ \( \Theta \), it is well-known that each Datalog program \( \Pi \) is equivalent to an infinite union \( \bigcup_{i \geq 1} \theta_i \) of (uniformly generated) CQs, and, therefore, checking if \( \Pi \subseteq \Theta \) amounts to checking if \( \bigcup_{i \geq 1} \theta_i \subseteq \Theta \). Restricting \( \Theta \) to belong to one of the tractable classes \( TW(k) \) and \( HW(k) \) reduces the complexity of containment when the left-hand side of the containment problem is a finite union of CQs. So far, however, the question of whether such restriction also helps alleviate the cost of checking containment of the infinite union \( \bigcup_{i \geq 1} \theta_i \) (and, thus, of \( \Pi \)) in \( \Theta \) has not been addressed. We study this question in depth in this paper.

Somewhat surprisingly, we prove that the situation is more delicate than expected:

1. First, we show that none of the tractable restrictions of UCQs that have been studied to date – save for \( TW(1) \) – helps to reduce the complexity of the Datalog containment problem. That is, checking whether \( \Pi \subseteq \Theta \), for \( \Pi \) a Datalog program and \( \Theta \) a UCQ, continues to be \( 2\text{EXPTIME} \)-complete even if (a) \( \Theta \) is of \( TW(2) \), or (b) \( \Theta \) is in \( HW(1) \), and, thus, acyclic. (This does not follow from known hardness results.)

2. Second, in order to obtain better complexity bounds we have to unveil a finer hierarchy of queries inside the class of acyclic UCQs. This hierarchy is
   \[
   AC_1 \subset AC_2 \subset \cdots \subset AC_k \subset \cdots,
   \]
   where \( AC_k \) denotes the class of UCQs \( \Theta \) in \( AC \) such that no two atoms in \( \Theta \) share more than \( k \) variables. We prove that checking \( \Pi \subseteq \Theta \), for \( \Theta \) in \( AC_k \), can be solved in \( \text{EXPTIME} \) (being complete for this class).

As corollaries, we obtain that checking \( \Pi \subseteq \Theta \) can be solved in \( \text{EXPTIME} \) when \( \Theta \) is an acyclic UCQ of fixed arity and when it is of treewidth 1. These results are optimal, since we prove that checking \( \Pi \subseteq \Theta \) is \( 2\text{EXPTIME} \)-complete, even if \( \Theta \) is a UCQ in \( HW(2) \) or \( TW(2) \) of fixed arity.

In the second part of the paper we switch to study the containment problem over graph databases [1, 35]. The analogue of UCQs in this context are the unions of conjunctive two-way regular path queries, UC2RPQs [7], that extend UCQs with the ability to check whether two nodes in a graph database are linked by a path that satisfies a regular condition. Calvanese et al. [8] proved that containment of Datalog in UC2RPQ is still decidable in \( 2\text{EXPTIME} \).

Evaluation of UC2RPQs is also NP-complete, but tractability can be obtained by considering the class \( ACR \) of acyclic UC2RPQs [3]. These are the UC2RPQs whose underlying CQs are acyclic (or, equivalently, of treewidth one, since \( AC = TW(1) \) over graph databases). Since containment of Datalog in UCQs in \( TW(1) \) can be solved in \( \text{EXPTIME} \), it is natural to study whether the restriction to acyclic UC2RPQs also helps alleviate the complexity of the containment problem in this context. We show that, again, the situation is more delicate than expected:

1. First, acyclicity in this case does not help to reduce the complexity. That is, containment of a Datalog program \( \Pi \) in a UC2RPQ \( \Gamma \) is \( 2\text{EXPTIME} \)-complete, even if \( \Gamma \) is a UC2RPQ in \( ACR \).

2. Again, in order to obtain better complexity bounds we need to unveil the finer structure of the class of acyclic UC2RPQs. In order to do that, we first identify a hierarchy
   \[
   ACR^1 \subset ACR^2 \subset \cdots \subset ACR^k \subset \cdots,
   \]
   where \( ACR^k \) is the class of UC2RPQs \( \Gamma \) in \( ACR \) in which each pair of variables is connected by at most \( k \) atoms, and then prove that \( \Pi \subseteq \Gamma \) can be solved in \( \text{EXPTIME} \) if \( \Gamma \) belongs to \( ACR^k \), for \( k \geq 1 \). Queries in \( ACR^1 \) have been previously studied in the literature under the name of strongly acyclic UC2RPQs [1].

This result is, in a sense, optimal, since containment of Datalog in UC2RPQs \( \Gamma \) of (hyper-)treewidth two is \( 2\text{EXPTIME} \)-complete, even if each pair of variables in \( \Gamma \) is connected by at most one atom.

**Organization.** We present preliminaries in Section 2 and tractable classes of UCQs in Section 3. Results on containment of Datalog in tractable UCQs are in Section 4, while Section 5 is devoted to the study of containment of Datalog in acyclic UC2RPQs. We finish in Section 6 with our concluding remarks.
2. PRELIMINARIES

Conjunctive queries. We assume familiarity with relational schemas $\sigma$ and databases $D$. Recall that a conjunctive query (CQ) over $\sigma$ is a logical formula in the $\exists, \wedge$-fragment of FO, i.e., a formula $\theta$ of the form

$$\exists y(R_1(x_1) \wedge \cdots \wedge R_m(x_m)),$$

where each $R_i(x_i)$ is an atom over $\sigma$ ($1 \leq i \leq m$). We write $\theta(\bar{x})$ to denote that $\bar{x}$ are the free variables of $\theta$, i.e., those that are not mentioned in the tuple $y$.

As usual, the semantics of CQs is defined in terms of homomorphisms. Formally, a homomorphism $h$ from a CQ $\theta$ of the form $\exists y(R_1(x_1) \wedge \cdots \wedge R_m(x_m))$ to a database $D$ over $\sigma$ is a mapping from the set of variables that appear in the $x_i$’s (for $1 \leq i \leq m$) to the elements of $D$, such that $R_i(h(\bar{x})) \in D$ for each $1 \leq i \leq m$. The evaluation of $\theta$ over $D$, denoted $\theta(D)$, consists of the set of all tuples $h(\bar{x})$, for $h$ a homomorphism from $\theta$ to $D$.

A union of CQs (UCQ) $\theta$ over $\sigma$ is a set $\{\theta_1(\bar{x}), \ldots, \theta_k(\bar{x})\}$ of CQs over $\sigma$ with the same free variables. We define $\Theta(D)$ to be $\bigcup_{1 \leq i \leq k} \theta_i(D)$, for each database $D$.

Datalog. Extending UCQs with recursion yields the Datalog language. Formally, a Datalog program $\Pi$ consists of a finite set of rules of the form

$$S(\bar{x}) \leftarrow R_1(\bar{x}_1), \ldots, R_m(\bar{x}_m),$$

where $S(\bar{x})$ and $R_i(\bar{x}_i)$ are atoms, for each $1 \leq i \leq m$, and each variable in $\bar{x}$ appears in some of the $\bar{x}_i$’s ($1 \leq i \leq m$). The atom $S(\bar{x})$ is the head of this rule, while its body is $R_1(\bar{x}_1), \ldots, R_m(\bar{x}_m)$. We denote by $\text{Rels}(\Pi)$ the set of relation symbols that appear in $\Pi$ and by $\text{IRel}(\Pi)$ the set of intensional relation symbols of $\Pi$, i.e., the set of symbols $S \in \text{Rels}(\Pi)$ such that there is an atom of the form $S(\bar{x})$ in the head of some rule of $\Pi$.

The semantics of a Datalog program $\Pi$ is defined as follows. Let $D$ be a database over $\text{Rels}(\Pi)$. Then $\mathcal{F}(D)$ is a database over $\text{Rels}(\Pi)$ that consists of all facts $S(\bar{t})$ such that $D$ contains a rule of the form $S(\bar{x}) \leftarrow R_1(\bar{x}_1), \ldots, R_m(\bar{x}_m)$ that satisfies $\bar{t}$ belongs to the evaluation of the CQ $\theta(\bar{x}) := \exists y(R_1(\bar{x}_1) \wedge \cdots \wedge R_m(\bar{x}_m))$ over $D$. We define the result of applying $\Pi$ to $D$ to be $\mathcal{F}^\infty(D) := \bigcup_{i \geq 0} \mathcal{F}^i(D)$, where

$$\mathcal{F}^0(D) := D \text{ and } \mathcal{F}^{i+1}(D) := \mathcal{F}(\mathcal{F}(D)) \cup \mathcal{F}(D),$$

for $i \geq 0$. Clearly, for every database $D$ we have $\bigcup_{i \geq 0} \mathcal{F}^i(D) = \bigcup_{0 \leq i \leq j} \mathcal{F}^i(D)$, for some $j \geq 0$, and, thus, $\mathcal{F}^\infty(D)$ is finite.

A Datalog program $\Pi$ is over the schema $\sigma$, if $\sigma = \text{Rels}(\Pi) \setminus \text{IRel}(\Pi)$ and there is a distinguished symbol $Q$ in $\text{IRel}(\Pi)$. If $\Pi$ is a database over $\sigma$, we define the evaluation $\Pi(D)$ of $\Pi$ over $D$ to be the set of tuples $\bar{t}$ such that $Q(\bar{t}) \in \mathcal{F}^\infty(D)$ (assuming that the interpretation in $D$ of each $Q \in \text{IRel}(\Pi)$ is empty).

EXAMPLE 1. This example is taken from [29]. Imagine a scenario in which we have compulsive consumers that buy everything they like, plus anything that is trendy in case they have bought something before. The shopping carts of these consumers can be defined by the Datalog program $\Pi$, that consists of rules

$$\text{buys}(x,y) \leftarrow \text{likes}(x,y)$$
$$\text{buys}(x,y) \leftarrow \text{trendy}(x), \text{likes}(z,y).$$

The program $\Pi$ is over the schema that consists of symbols $\text{trendy}$ and $\text{likes}$. The distinguished symbol is $\text{buys}$.

Containment of queries. Let $q_1$ and $q_2$ be queries over the same schema $\sigma$ in one of the query languages we introduced before. Then $q_1$ is contained in $q_2$, denoted $q_1 \subseteq q_2$, if and only if $q_1(D) \subseteq q_2(D)$ for every database $D$ over $\sigma$. Notice that $q_1$ is equivalent to $q_2$ iff $q_1 \subseteq q_2$ and $q_2 \subseteq q_1$.

Let $C_1$ and $C_2$ be two classes of queries, e.g., CQs, UCQs, Datalog, or some of the classes we introduce in the paper. We define the containment problem of $C_1$ in $C_2$ as follows:

<table>
<thead>
<tr>
<th>PROBLEM</th>
<th>CONT(C_1, C_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT</td>
<td>Queries $q_1 \in C_1$ and $q_2 \in C_2$ over same schema $\sigma$.</td>
</tr>
<tr>
<td>QUESTION</td>
<td>Is $q_1 \subseteq q_2$?</td>
</tr>
</tbody>
</table>

We also study the complexity of this problem for schemas of fixed arity. Formally, let $c$ be a positive integer. We then write $\text{CONT}_c(C_1, C_2)$ to denote the restriction of the problem $\text{CONT}(C_1, C_2)$ to inputs in which the arity of $\sigma$ is at most $c$.

Containment of UCQs. It has long been known that containment and evaluation of CQs are equivalently hard. In fact, let $\theta(\bar{x})$ and $\theta'(\bar{x})$ be CQs over $\sigma$ and assume that $\theta$ is of the form $\exists y(R_1(\bar{x}_1), \ldots, R_m(\bar{x}_m))$. Let $D_\theta$ be the canonical database of $\theta$, i.e.,

$D_\theta := \{R_i(\bar{x}_i) \mid 1 \leq i \leq m\}.$

It follows from the seminal work of Chandra and Merlin [10] that $\theta \subseteq \theta'$ iff $\bar{x} \in \theta'(D_\theta)$. Furthermore, Sagiv and Yannakakis proved that if $\theta$ and $\theta'$ are UCQs, then $\theta \subseteq \theta'$ iff for every CQ $\theta \in \Theta$ there exists a CQ $\theta' \in \Theta'$ such that $\theta \subseteq \theta'$ [31]. Since CQ evaluation is NP-complete [10], we obtain the following:

**THEOREM 1.** [10, 31] $\text{CONT}(\text{UCQ}, \text{UCQ})$ is NP-complete.

Containment of Datalog in UCQ. As opposed to the case of UCQs, checking containment for Datalog programs is undecidable [33]. This has motivated the search for relevant restrictions of the problem that can be effectively decided (ideally at a reasonable computational cost). We study in this paper one of the most important of such decidable restrictions: containment of Datalog in UCQs [12], $\text{CONT}(\text{Datalog}, \text{UCQ})$. In full generality, this is a computationally expensive problem, even for schemas of fixed arity.

**THEOREM 2.** [12] $\text{CONT}(\text{Datalog}, \text{UCQ})$ is in 2EXPtime. There is $c \geq 1$ such that the problem $\text{CONT}(\text{Datalog}, \text{UCQ})$ is 2EXPtime-complete.

**EXAMPLE 2.** Consider the compulsive consumers program $\Pi$, from Example 1. It can be shown to be contained (and, in fact, it is equivalent to) the following UCQ:

$$\{\text{likes}(x,y), \exists z(\text{trendy}(x) \wedge \text{likes}(z,y))\}.$$ 

3. TRACTABLE CONTAINMENT FOR UCQ

While in general UCQ containment is an NP-complete problem, we obtain tractability by imposing that UCQs $\Theta'$ in the right-hand side of the containment problem belong to a tractable class; that is, to a class $\mathcal{C}$ of UCQs for which there is a polynomial time algorithm that given a UCQ $\Theta(\bar{x})$ in $\mathcal{C}$, a database $D$, and a tuple $\bar{t}$ of the same arity than $\bar{x}$, checks whether $\bar{t} \in \Theta(D)$. Formally:

**PROPOSITION 1.** [10, 31] Let $\mathcal{C}$ be a tractable class of UCQs. Then $\text{CONT}(\text{UCQ}, \mathcal{C})$ is in PTIME.
Due to a myriad of papers in the last two decades, we have by now a very good understanding of which classes of UCQs are tractable. These include classes of bounded treewidth [14], hypertreewidth [24], generalized hypertreewidth [24], fractional hypertreewidth [25], etc. We concentrate on the first two, which are defined next.

**UCQs of bounded treewidth.** An important tractable class of UCQs can be obtained by restricting the treewidth of the Gaifman graph of queries [14]. Formally, the Gaifman graph of a CQ $\theta$ of the form $\exists y (R_1(x_1), \ldots, R_m(x_m))$ is the undirected graph $G_\theta$ whose nodes are the variables of $\theta$, and there is an edge in $G_\theta$ from variable $x$ to $x'$ if there is $1 \leq i \leq m$ such that both $x$ and $x'$ appear in the tuple $\bar{x}_i$.

A tree decomposition of an undirected graph $G = (V, E)$ is a pair $(T, \lambda)$, where $T$ is a tree and $\lambda : T \to 2^V$, that satisfies the following:

1. For each $v \in V$ the set $\{t \in T \mid v \in \lambda(t)\}$ is a connected subset of $T$.
2. Each edge of $E$ is contained in one of the sets $\lambda(t)$, for $t \in T$.

The width of $(T, \lambda)$ is $\max \{|\lambda(t)| \mid t \in T\} - 1$. The treewidth of $G$ is the minimum width of its tree decompositions. Intuitively, the treewidth of $G$ measures its tree-likeness. Notice that $G$ is acyclic iff it is of treewidth one.

The treewidth of the UCQ $\Theta$ is the maximum treewidth of $G_\theta$, for $\theta \in \Theta$. We denote by $\text{TW}(k)$ the class of UCQs of treewidth at most $k$, for $k \geq 1$.

**Example 3.** The CQ $E(x_1, x_2) \land \cdots \land E(x_{n-1}, x_n)$ is in $\text{TW}(1)$, for each $n \geq 3$. In fact, its Gaifman graph is a path, and, thus, acyclic. Adding $E(x_1, x_n)$ increases the treewidth to two. Adding all atoms of the form $E(x_i, x_j)$, for $1 \leq i < j \leq n$, yields a CQ whose Gaifman graph is a clique of size $n$. The treewidth of this CQ is $n - 1$.

It follows from [14] (see also [18]) that each layer of the hierarchy $\text{TW}(1) \subset \text{TW}(2) \subset \cdots \subset \text{TW}(k) \subset \cdots$ is tractable. From Proposition 1 we thus have:

**Theorem 3.** Let $k \geq 1$. Then $\text{Cont}(\text{UCQ}, \text{TW}(k))$ can be solved in $\text{Ptime}$.

**UCQs of bounded hypertreewidth.** The notion of treewidth is defined in terms of the Gaifman graph of a CQ, but this is too restrictive when the arity of the schemas is not fixed in advance. In order to overcome this limitation, Gottlob et al. [24] proposed studying syntactic restrictions of the class of CQs based on properties of the underlying hypergraph of queries. The analogue of treewidth in this context is the notion of hypertreewidth, which, like the former, leads to tractability of query evaluation. In particular, each layer of the hierarchy $\text{HW}(1) \subset \text{HW}(2) \subset \cdots \subset \text{HW}(k) \subset \cdots$, where $\text{HW}(k)$ is the class of UCQs of hypertreewidth at most $k$, is tractable [24]. We skip the definition of the $\text{HW}(k)$’s since it is rather technical and not crucial for our purposes (the interested reader can find the definition in [24]).

Since the $\text{HW}(k)$’s are tractable, we obtain from Proposition 1 the tractability of $\text{Cont}(\text{UCQ}, \text{HW}(k))$:

**Theorem 4.** Let $k \geq 1$. Then $\text{Cont}(\text{UCQ}, \text{HW}(k))$ can be solved in $\text{Ptime}$.

It is important for us to notice that the lowest level $\text{HW}(1)$ of the hierarchy coincides with the well-known class $\text{AG}$ of acyclic UCQs [36], which admits a simple definition. A UCQ $\Theta$ is in $\text{AG}$ iff each CQ $\theta \in \Theta$ can be represented as a join tree [4]. The latter means that there is a tree $T$ whose nodes are the atoms of $\theta$, such that for each variable $x$ in $\theta$ it is the case that the set of nodes in which $x$ is mentioned is a connected subset of $T$.

The notions of bounded treewidth and acyclicity are incomparable. For instance, consider the class $\mathcal{C} = \{\theta_n \mid n \geq 2\}$, where $\theta_n := \bigwedge_{1 \leq i \leq n} E(x_i, x_i) \land T_n(x_1, \ldots, x_n)$. Each CQ $\theta_n \in \mathcal{C}$ is acyclic; this is witnessed by the join tree $T$ whose root $r$ corresponds to the atom $T_n(x_1, \ldots, x_n)$ and there is a child of $r$ for each atom of the form $E(x_i, x_j)$, for $1 \leq i < j \leq n$. On the other hand, the treewidth of the CQs in $\mathcal{C}$ is not bounded by a constant. Consider now the class $\mathcal{C}$ of queries $\{E(x_1, x_2) \land \cdots \land E(x_{n-1}, x_n) \land R(x_1, x_n) \mid n \geq 3\}$. Each CQ in $\mathcal{C}$ is in $\text{TW}(2)$, but no such CQ is acyclic.

**4. CONTAINMENT OF DATALOG IN TRACTABLE CLASSES OF UCQs**

It is well-known that each Datalog program $\Pi$ can be expressed as an infinite union $\bigcup_{i \geq 1} \theta_i^{(1)}$ of CQs. The $\theta_i^{(1)}$’s are called the expansions of $\Pi$ (see, e.g., [29]). Therefore, checking containment of $\Pi$ in a UCQ $\Theta$ reduces to checking containment of the infinite UCQ $\bigcup_{i \geq 1} \theta_i^{(1)}$ in $\Theta$. Restricting $\Theta$ to belong to one of the tractable classes $\text{TW}(k)$ and $\text{HW}(k)$ reduces the complexity of containment when the left-hand side of the containment problem is a finite union of CQs (Theorems 3 and 4). On the other hand, it is not known whether such restriction also helps alleviate the cost of checking containment of the infinite union $\bigcup_{i \geq 1} \theta_i^{(1)}$ (and, thus, of $\Pi$) in $\Theta$. We show in this section that the situation is much more delicate than expected. But before doing so, it is important to explain how expansions can be represented using expansion trees, and how containment of Datalog programs in UCQs can be checked using those expansion trees.

**Expansion trees.** We quickly recall the notion of (unfolding) expansion tree from [12]. Let $\Pi$ be a Datalog program over $\sigma$ with distinguished symbol $Q$. Then:

1. The nodes of an expansion tree of $\Pi$ are labeled with instances of rules in $\Pi$.
2. The root of an expansion tree of $\Pi$ is labeled with a rule whose head is of the form $Q(\bar{x})$.
3. For every node $u$ of an expansion tree such that $u$ is labeled $\rho$ and $R_1(x_1), \ldots, R_t(\bar{x}_t)$ are the atoms of the body of $\rho$ over the schema $\text{IRels}(\Pi)$, it is the case that $u$ has exactly $t$ children $u_1, \ldots, u_t$ in the expansion tree and each $u_j$ is labeled with an instance $\rho_j$ of a rule in $\Pi$ whose head is $R_j(\bar{x}_j)$, for $1 \leq j \leq t$.

Therefore, leaves of expansion trees have to be labeled with instances of rules of $\Pi$ with no intensional atoms.

4. For every node $u$ of an expansion tree labeled $\rho$, it is the case that every variable $y$ mentioned in the body of $\rho$ either appears in the head of $\rho$ or it does not appear in any node above $u$ in the expansion tree.

This is a technical condition that ensures that when creating children of nodes in expansion trees in order to unify intensional atoms, only “fresh” instances of rules in $\Pi$ can be used.
Clearly, the number of different expansion trees of a program $\Pi$ might be infinite (in particular, when $\Pi$ is recursive).

Each expansion tree $\tau$ represents a CQ $\theta_{\tau}$ over $\sigma$: this is obtained by taking the conjunction of all atoms over $\sigma$ that label the nodes of $\tau$. Assume that the root of $\tau$ is labeled with a rule whose head is $Q(\bar{x})$. Then $\bar{x}$ is the tuple of free variables of $\theta_{\tau}$. The CQs of the form $\theta_{\tau}$, for $\tau$ an expansion tree of $\Pi$, are precisely the expansions of $\Pi$, i.e. the CQs $\theta^\ell_i$ such that $\Pi$ is equivalent to $\bigcup_{i \geq 1} \theta^\ell_i$ [12]. It follows that $\Pi$ is contained in a UCQ $\Theta$ iff for each expansion tree $\tau$ of $\Pi$ with associated CQ $\theta_{\tau}(\bar{x})$ there is a CQ $\theta' \in \Theta$ such that $\theta_{\tau} \subseteq \theta'$, or equivalently, $\bar{x} \in \theta'(D_{\bar{a}_{\bar{x}}})$.

This last condition is usually rephrased in terms of the existence of a containment mapping from $\theta'$ to $\theta$ [12], i.e. a mapping $\mu$ from the variables of $\theta'$ to the variables that label the atoms of $\tau$, such that (i) $\mu$ is the identity over $\bar{x}$, and (ii) for each atom $R(y)$ in $\theta'$ it is the case that $R(\mu(y))$ appears in the label of a node of $\tau$.

Clearly, the number of different expansion trees of a program $\Pi$ is $2^{|\Pi|}$, and the number of different expansion trees of a program $\Pi$ is $2^{|\Pi|}$, and the number of different expansion trees of a program $\Pi$ is $2^{|\Pi|}$.

### Proposition 2

Let $\Pi$ be a Datalog program and $\Theta$ a UCQ. Then $\Pi \subseteq \Theta$ iff for every expansion tree $\tau$ of $\Pi$ there is a CQ $\theta_{\tau} \in \Theta$ and a containment mapping from $\theta_{\tau}$ to $\tau$.

Several of our proofs rely on this characterization.

## 4.1 Tractable UCQs are not enough

We show that tractable restrictions on UCQs do not help, in general, to reduce the complexity of $\text{Cont}$(Datalog,UCQ). In particular, we prove that 2EXPTIME-hardness for checking containment of a Datalog program $\Pi$ in a UCQ $\Theta$ holds even if $\Theta$ is in HW(1) = AC, or it is in TW(2). These negative results are quite resilient: They hold even for UCQs in TW(2) or HW(2) over schemas of fixed arity.

### Theorem 5

The following problems are 2EXPTIME-hard:

1. $\text{Cont}$(Datalog,AC).
2. $\text{Cont}_2$(Datalog,HW(2)).
3. $\text{Cont}_2$(Datalog,TW(2)).

None of these results follows from existing hardness results for $\text{Cont}$(Datalog,UCQ) in the literature [5, 12]. Instead, we have to carefully refine the techniques in [12] to obtain these stronger lower bounds. Next, we sketch the proof of 2EXPTIME-hardness for $\text{Cont}$(Datalog,AC).

**Proof (Sketch):** We reduce from the following 2EXPTIME-complete problem: Given an alternating Turing machine $M$ and a positive integer $n$, decide whether $M$ accepts the empty tape using $2^n$ space. Recall that alternating Turing machines $M$ have existential and universal states. We assume w.l.o.g. that (1) the initial state of $M$ is existential, (2) $M$ always alternates between existential and universal states, and (3) every configuration of $M$ has two possible successors, a left successor and a right successor, defined by two deterministic transitions functions $\delta$ and $\delta'$. An accepting computation of $M$ is a tree of configurations, where each configuration is a successor of its parent, a universal configuration has both of its successors as children, and all leaves are accepting.

A configuration of $M$ can be described as a string of length $2^n$. The symbols of the string are either symbols of the alphabet or composite symbols. A composite symbol is a pair $(q, e)$, where $q$ is a state of $M$ and $e$ is in the alphabet of $M$. Intuitively, a symbol $(q, e)$ indicates that $M$ is in state $q$ and is scanning the symbol $e$. It is well known that the successor relation between configurations depends only on local constraints: we can associate with $\delta$ two ternary relations $I'$, $F'$ and a 4-ary relation $B'$ on symbols that characterize the left successor relation. If $\bar{a} = a_1, \ldots, a_m$ and $\bar{b} = b_1, \ldots, b_m$ are two configurations, then $\bar{b}$ is a left successor of $\bar{a}$ if $(a_1, a_2, b_1) \in I'$, $(a_{m-1}, a_m, b_m) \in F'$ and $(a_{i-1}, a_i, a_{i+1}, b_i) \in B'$ for each $1 < i < m$. Analogously, we associate with $\delta'$ the relations $I''$, $F''$ and $B''$.

We construct a Datalog program $\Pi$ and a UCQ $\Theta$ in AC that encode accepting computations of $M$. The expansions of $\Pi$ will correspond to configuration trees, and each disjunct in $\Theta$ will detect a particular error that prevents an expansion from being an accepting computation. Thus, if an expansion $\tau$ does not correspond to an accepting computation, we will have that $\tau \subseteq \Theta$. Therefore, $\Pi \subseteq \Theta$ if and only if the machine $M$ does not accept the empty tape using $2^n$ space. In our construction, we need to compare corresponding positions in successive configurations in order to detect transition errors. This is achieved by identifying each position in a configuration with an $n$-bit address.

**Schema.** The schema of $\Pi$ and $\Theta$ consists of a symbol $A$ of arity $n + 8$, a unary symbol $Start$ and a unary symbol $\bar{Q}$, for each possible symbol $s$. The intuition behind the predicate $A$ is as follows: (1) the first two arguments of $A$ act as the constants 0 and 1, (2) the third and fourth arguments of $A$ link successive addresses, (3) the next $n$ arguments of $A$ encode the address, (4) the next three arguments of $A$ link successive configurations, and (5) the last argument indicates whether the current configuration is existential or universal.

**Program II.** The program $\Pi$ contains an intentional relation symbol $B$ of arity $n + 7$ and a 0-ary distinguished symbol $C$. Expansion trees of $\Pi$ correspond to computation trees of $M$. In an expansion tree, the predicate $B$ propagates the information while it generates atoms of the form $A(x, y, z, z', a_1, \ldots, a_n, u, v, w, t), Q_s(z)$. Intuitively, this represents that, in the current configuration, the position at address $a_1 \ldots a_n$ contains the symbol $s$.

For $1 \leq i \leq n$, the program $\Pi$ contains the following rules:

$$
B(x, y, z, a_1, \ldots, a_n, u, v, w, t) \\
\leftarrow B(x, y, z, a_1, \ldots, a_{i-1}, x, a_{i+1} \ldots, a_n, u, v, w, t).
$$

$$
B(x, y, z, a_1, \ldots, a_n, u, v, w, t) \\
\leftarrow B(x, y, z, a_1, a_{i-1}, y, a_{i+1} \ldots, a_n, u, v, w, t).
$$

Intuitively, each unfolding of these rules modifies the $i$-th address bit: $x$ encodes the bit 0, and $y$ the bit 1. For each symbol $s$, $\Pi$ contains the rule:

$$
B(x, y, z, a_1, \ldots, a_n, u, v, w, t) \\
\leftarrow A(x, y, z', a_1, \ldots, a_n, u, v, w, t), \\
Q_s(z), B(x, y, z', a_1, \ldots, a_n, u, v, w, t).
$$

These rules determine the symbols in the configurations.

To encode transitions from configuration to configuration, we have to check whether the source configuration is existential or universal. For existential configurations we have rules of the form:

$$
B(x, y, z, a_1, \ldots, a_n, u, v, w, x) \\
\leftarrow A(x, y, z', a_1, \ldots, a_n, u, v, w, x), \\
Q_s(z), B(x, y, z', a_1, \ldots, a_n, u', v, w', y).
$$

$$
B(x, y, z, a_1, \ldots, a_n, u, v, w, x) \\
\leftarrow A(x, y, z', a_1, \ldots, a_n, u, v, w, x), \\
Q_s(z), B(x, y, z', a_1, \ldots, a_n, u', v, w, y).
$$
Note that \( u \) moves either one or two positions to the right in the \( B \) predicate. A one-position movement corresponds to a transition to a left successor, while a two-position movement corresponds to a transition to the right successor.

Equivalently, for universal configurations we have rules:

\[
B(x, y, z, x_1, \ldots, x_n, u, v, w, y) \\
\quad \leftarrow A(x, y, z, x', a_1, \ldots, a_n, u, v, w, y), Q_s(z) \\
B(x, y, x', a_1, \ldots, a_n, u', u', w', x), \\
B(x, y, z', a_1, \ldots, a_n, u', u', v, u, x).
\]

To encode the start of the computation we use the rule:

\[
C \leftarrow \text{Start}(z), B(x, y, z, x, \ldots, x, u, v, w, x).
\]

To encode the end of the computation, we use rules:

\[
B(x, y, z, a_1, \ldots, a_n, u, v, w, t) \\
\quad \leftarrow Q_s(z), A(x, y, z, z, a_1, \ldots, a_n, u, v, w, t),
\]

for symbols \( s = (q, e) \) such that \( q \) is an accepting state.

The UCQ \( \Theta \). Now we how show how to detect errors in the expansion trees of \( \Pi \) using a Boolean acyclic UCQ \( \Theta \). Each disjunct of \( \Theta \) detects a particular type of error. There are first some simple errors that prevent the expansion tree from being a configuration tree: (a) the first address is not \( 0, \ldots, 0 \), (b) there are two consecutive addresses \( \overline{a} \) and \( \overline{b} \) such that \( \overline{b} \neq \overline{a} + 1 \) (mod 2\( n \)), (c) a configuration does not change when the address is \( 1, \ldots, 1 \), and (d) a configuration changes when the address is \( 1, \ldots, 1 \). For each one of them we can easily adapt techniques from [12] and build an acyclic CQ that detects it.

We have so far ensured that the expansion tree is a configuration tree. Now we have to ensure that the tree is a valid computation of the machine \( M \). In order to force the first configuration to be the initial configuration, we can again apply techniques from [12]. On the other hand, to detect errors in the transitions \( \delta' \) and \( \delta'' \) we cannot directly apply those techniques. This is because the CQs constructed in [12] to detect those errors are not acyclic. We use, instead, the following idea.

For each \((a, b, c, d) \notin B'\) we add the CQ that is defined by the conjunction of every atom in the set:

\[
\Phi(a, b, c, d) := \{A(x, y, z_1, z_2, \overline{a}_1, u, v, w, u_1), Q_s(z_1), A(x, y, z_2, z_3, \overline{a}_2, u, v, w, u_2), Q_s(z_2), A(x, y, z_3, z_4, \overline{a}_3, u, v, w, u_3), Q_s(z_3), A(x, y, z, z', \overline{a}_4, u', u', w', u_4), Q_s(z)\}.
\]

The pattern of the \( z_i \)'s variables and the \( u, v, w \) variables ensures that the first three atoms are mapped in three successive positions in the same configuration \( E \). The pattern \("u', u', w'"\) ensures that the last atom is mapped to the left successor of \( E \). The reuse of \( \overline{a}_2 \) ensures that the second and the last atom are mapped to the same address. Note that each one of these queries is actually acyclic, as witnessed by the join tree in which the first three \( A \)-atoms form a path and the last \( A \)-atom is a child of the second one. We define similar CQs in \( AC \) for each \((a, b, c) \notin F'\), for each \((a, b, c) \notin F''\), and for detecting errors in \( \delta' \).

It is not hard to prove now that \( \Pi \subseteq \Theta \) iff \( M \) does not accept the empty tape in space \( 2^n \). Since the construction of \( \Pi \) and \( \Theta \) can be carried out in polynomial time, it follows that \( \text{Cont(Datalog,AC)} \) is 2EXPTIME-hard. \( \Box \)

### 4.2 Better bounds for queries in \( AC_k \)

Since restricting to a tractable class of UCQs the right-hand side of the problem \( \text{Cont(Datalog,UCQ)} \) does not help, in general, to reduce its computational cost, we look for further restrictions on UCQs that yield better bounds.

A big source of complexity for \( \text{Cont(Datalog,AC)} \) is the existence of pairs of atoms in CQs that share an unbounded number of variables (in the proof of Theorem 5 this is witnessed, for instance, in the CQs of the form \( \delta(a, b, c, d) \), for \((a, b, c, d) \notin B'\)). By restricting this parameter we unveil a new hierarchy inside the class of acyclic UCQs, that consists of the acyclic UCQs with a bounded number of common variables between atoms. We prove that containment of Datalog in any of the levels of this hierarchy can be solved in EXPTIME (and it is actually complete for this class).

Formally, let us define \( \text{AC}_k \), for \( k \geq 1 \), to be the class of queries \( \Theta \) in \( AC \) such that no two distinct atoms in a CQ in \( \Theta \) share more than \( k \) variables. Clearly:

\[
\text{AC}_1 \subset \text{AC}_2 \subset \cdots \subset \text{AC}_k \subset \cdots \subset AC.
\]

**Example 4.** The CQ \( \bigwedge_{1 \leq i \leq n} E(x, x_{i+1}) \), for \( n \geq 2 \), is in \( AC_1 \). The CQ \( \bigwedge_{1 \leq i < j \leq n} E(x_i, x_j) \land T_n(x_1, \ldots, x_n) \), for \( n \geq 2 \), is in \( AC_2 \).

**Theorem 6.** Let \( k \geq 1 \). Then \( \text{Cont(Datalog,AC}_k \text{)} \) is EXPTIME-complete.

**Proof (Sketch):** We only sketch the upper bound. We assume familiarity with standard (one-way) nondeterministic tree automata [32], INTA, and two-way alternating tree automata [34], 2ATA, which extend INTA with both upward moves and alternation.

**Overall idea.** It is known that each expansion tree of a Datalog program \( \Pi \) can be represented as a ranked tree over a finite alphabet \( \Sigma_1 \). Such representations are called the *proof trees* of \( \Pi \). The language of all proof trees of \( \Pi \) can be defined by a INTA \( A_{\Pi} \) over \( \Sigma_1 \), such that \( A_{\Pi} \) can be constructed in single exponential time from \( \Pi \) [12]. Consider first the case when \( \Theta \) is an arbitrary UCQ (i.e., not necessarily in \( \text{AC}_k \)). Then it is possible to construct in single exponential time from \( \Pi \) and a UCQ \( \Theta \), a INTA \( A_{\Pi}^{\Theta} \) that accepts precisely the proof trees \( \nu \) of \( \Pi \) such that there is a CQ \( \theta \in \Theta \) and a containment mapping from \( \theta \) to the expansion tree \( \tau \) represented by \( \nu \) [12]. (Both \( A_{\Pi} \) and \( A_{\Pi}^{\Theta} \) can be of exponential size). Thus, in order to check if \( \Pi \subseteq \Theta \), it is sufficient to check whether the language defined by the INTA \( A_{\Pi} \) is contained in the one defined by \( A_{\Pi}^{\Theta} \). The latter can be done in double exponential time in the size of \( \Pi \) and \( \Theta \) [32].

In our case, that is, when \( \Theta \) belongs to \( \text{AC}_k \), we can follow a slightly different approach and check containment of \( \Pi \) in \( \Theta \) in single exponential time. We start again by constructing in \( \text{EXPTIME} \) the INTA \( A_{\Pi} \) over \( \Sigma_1 \) that defines the set of proof trees of \( \Pi \). The difference is that we now construct in exponential time from \( \Pi \) and a 2ATA \( B_1^{\Pi} \) of polynomial size\(^1\), that accepts all trees \( T \) over \( \Sigma_1 \) such that there exists a CQ \( \theta \in \Theta \) and a strong containment mapping from \( \theta \) to \( T \). Notice that, in opposition to \( A_{\Pi}^{\Theta} \), the 2ATA \( B_1^{\Pi} \) may accept trees over \( \Sigma_1 \) that are not proof trees of \( \Pi \).

Checking whether \( \Pi \subseteq \Theta \) reduces to check if the language defined by \( A_{\Pi} \) is contained in the one defined by \( B_1^{\Pi} \). In order to

---

\(^1\)This means that even if \( \Sigma_1 \) is exponential, both the set of states \( S \) of \( B_1^{\Pi} \) and the sets of the form \( \delta(s, a) \), where \( \delta \) is the transition function of \( B_1^{\Pi} \), \( s \in S \) and \( a \in \Sigma_1 \), are of polynomial size.
do the latter, we first compute from $B_{\Theta}^0$ a 1NTA that defines the
complement of $B_{\Theta}^0$, and then check for nonemptiness the intersec-
tion of this 1NTA with $A_{\Theta}$. This tells us whether $\Pi \subseteq \Theta$. The
result now follows from EXPTIME being closed under complement and
the following facts: (1) A 1NTA that defines the complement of $B_{\Theta}^0$
can be constructed in single exponential time from $\Pi$ and $\Theta$
[17], and (2) checking if the language defined by the intersection
of this 1NTA with $A_{\Theta}$ is nonempty can be solved in time exponential
in the size of $\Pi$ and $\Theta$.

Technical details of the construction. Proof trees of $\Pi$ describe
expansion trees of $\Pi$ using a finite number of labels. Formally, let $r$ be a rule of $\Pi$ and $\nu(r)$ be the number of variables in $r$. We
define $\nu(\Pi)$ to be twice the maximum of $\nu(r)$, for all rules $r$ in
$\Pi$. Let $vars(\Pi)$ be a set of variables of cardinality $\nu(\Pi)$. A proof
tree $\nu$ of $\Pi$ is simply an expansion tree that does not satisfy the last
condition of the definition (that is, variables in $\nu$ can be reused),
and all the variables used in $\nu$ come from $vars(\Pi)$.

In an expansion tree, when we “unfold” a node we take a “fresh”
copy of a rule $r$ in $\Pi$. In a proof tree, we take instead an instance
of $r$ over $vars(\Pi)$. Since the number of variables in $vars(\Pi)$ is
twice the number of variables in any rule of $\Pi$, we can instantiate
the variables in the body of $r$ that do not appear in its head by
variables different from those in the head. Notably, the set of proof
trees of $\Pi$ can be described by a tree automaton $A_{\Pi}$. Formally, let $\Sigma_\Pi$ be the alphabet that consists of all instances of rules of $\Pi$ that
can be formed with variables from $vars(\Pi)$.

**Lemma 1.** [12] There exists an EXPTIME algorithm that, given
a Datalog program $\Pi$, constructs a 1NTA $A_{\Pi}$ over $\Sigma_\Pi$ such that
the language defined by $A_{\Pi}$ is precisely the set of proof trees of $\Pi$.

Before following with the construction, it is necessary to under-
stand when two occurrences of the same variable in a proof tree $\nu$
denote the same element in the expansion tree represented by $\nu$.
Let $T$ be a tree over $\Sigma_\Pi$ (e.g., $T$ might be a proof tree of $\Pi$). Assume
that $n_1$ and $n_2$ are nodes of $T$ with lowest common ancestor $n$,
and $x_1$ and $x_2$ are variables in the same variable $x$ from $vars(\Pi)$
in $n_1$ and $n_2$, respectively. Then $x_1$ and $x_2$ are connected in $T$, if
there is an occurrence of $x$ in the head of the label of every node,
except perhaps for $n$, in the simple path from $n_1$ to $n_2$ in $T$. An
occurrence $x_1$ of $x$ in $T$ is distinguished, if it is connected to an
occurrence of $x$ in the head of the label of the root of $T$.

Containment of $\Pi$ in $\Theta$ can be rephrased in terms of the exis-
tence of strong containment mappings from the CQs in $\Theta$ to trees
$T$ over $\Sigma_{\Pi}$ [12]. Let $\theta(\bar{x})$ be a CQ in $\Theta$ and $T$ a tree over $\Sigma_{\Pi}$
such that the head of the rule that labels the root of $T$ is $Q(\bar{x}^\prime)$,
for $|\bar{x}| = |\bar{x}^\prime|$. A strong containment mapping $\mu$ from $\theta(\bar{x})$ to $T$
maps occurrences of variables in $\theta$ to occurrences of variables in
$vars(\Pi)$, in such a way that the following holds: (i) if $y_1$ and $y_2$
are two occurrences of the same variable in $\theta$, then $\mu(y_1)$ and $\mu(y_2)$
are connected occurrences of the same variable in $T$, (ii) for each occurrence $x$ of $x$ in $T$, if $x$ is the case that $\mu(x)$ is a distinguished occurrence in $T$ of the $i$-th variable $x_i$ of $\bar{x}^\prime$, and (iii) for each atom in $\theta$ of the form $R(\bar{y})$, we have that $R(\mu(\bar{y}))$ appears in the label of some node of $T$.

Intuitively, a strong containment mapping from $\theta$ to a proof tree $\nu$
enforces that different occurrences of the same variable $y$ in $\theta$ are
mapped to the same element in the expansion tree $\tau$ represented
by $\nu$. With this idea in mind, it is not hard to prove that there is a
strong containment mapping from $\theta$ to $\nu$ if there is a containment
mapping from $\theta$ to $\tau$. We obtain the following from Proposition 2:

**Lemma 2.** [12] Let $\Pi$ be a Datalog program and $\Theta$ a UCQ.
Then $\Pi \subseteq \Theta$ iff for every proof tree $\nu$ of $\Pi$ there is a CQ $\theta \in \Theta$ and a strong containment mapping from $\theta$ to $\nu$.

In order to complete the proof of the theorem, it will be sufficient
to prove Lemma 3 below. This lemma states that, if $\Theta \in AC_k$, for
$k \geq 1$, it is possible to construct in single exponential time a 2ATA
$B_{\Theta}^0$, whose size is polynomial in the size of $\Pi$ and $\Theta$, such
that the language defined by $B_{\Theta}^0$ is the set of trees $T$ over $\Sigma_{\Pi}$
for which there is a CQ $\theta \in \Theta$ and a strong containment mapping
from $\theta$ to $T$.

**Lemma 3.** Let $k \geq 1$. There is an EXPTIME algorithm that,
given a Datalog program $\Pi$, and a UCQ $\Theta \in AC_k$, constructs a
2ATA $B_{\Theta}^0$, whose size is polynomial in the size of $\Pi$ and $\Theta$, such
that the language defined by $B_{\Theta}^0$ is the set of trees $T$ over $\Sigma_{\Pi}$
for which there is a CQ $\theta \in \Theta$ and a strong containment mapping
from $\theta$ to $T$.

The 2ATA $B_{\Theta}^0$ looks for a strong containment mapping from
some CQ in $\Theta$ to the tree $T$. In order to do this, it first guesses
a CQ $\theta \subseteq \Theta$, and then operates in a top-down fashion over the
join tree of $\theta$ (which exists since $\theta$ is acyclic). At each point, $B_{\Theta}^0$ is in a state that summarizes the following information: the atom $A$ of
the join tree of $\theta$ that is being scanned (which is the one that $B_{\Theta}^0$
is currently trying to match in $T$), and an assignment $M$ from
the variables of $\theta$ that have already been mapped to the variables of
$T$ (which restricts the possible matchings for $A$ in $T$, since some of
the variables of $A$ may belong to the domain of $M$). A transition
results in either the mapping of the current atom or a movement to
an adjacent node in $T$.

The 2ATA $B_{\Theta}^0$ is defined as $\cup_{\theta \in \Theta} B_{\Theta}^0$. We define 2ATAs of the form $B_{\Theta}^0$ below. Let $J$ be a join tree of $\theta$. We write $J$ also to refer to the set of atoms of $\theta$. Assume that $x_1, \ldots, x_m$ are the distinguished
variables of $\theta$. For an atom $A \subseteq J$, we define $dvar(A)$ to be the set of
indices in $\{1, \ldots, m\}$ such that $x_i$ appears in $A$.

The state set $S$ of $B_{\Theta}^0$ consists of two types of states: (1) atom
states in $J \times M_k$, where $M_k$ consists of all partial mappings from
the variables of $\theta$ to $vars(\Pi)$ whose domain has at most $k$ elements,
and (2) variable states in the set $\{1, \ldots, m\} \times vars(\Pi)$ (recall
that $m$ is the number of distinguished variables in $\theta$). There
is also an accepting state accept. The initial state of $B_{\Theta}^0$ is the atom
state $(A, \emptyset)$, where $A$ is the root of $J$ and $\emptyset$ denotes the empty
mapping.

Since $B_{\Theta}^0$ is a 2ATA, the transition function of $B_{\Theta}^0$ maps each
pair $(s, r)$, where $s \in S$ and $r \in \Sigma_{\Pi}$, to a positive formula $\phi$ in
DNF over the set $\{-1, 0, 1, \ldots, \ell\} \times S$, where $\ell$ is the maximum
number of intensional atoms in a rule of $\Pi$. Intuitively, when $B_{\Theta}^0$ is in a state $s$ reading symbol $r$, it first chooses a conjunction in $\phi$,
and then for each literal in this conjunction of the form $\langle c, s' \rangle$, for $c \in \{-1, 0, 1, \ldots, \ell\}$ and $s' \in S$, it launches a new copy of itself
in the direction suggested by $c$ starting in state $s'$.

The transition function $\delta$ of $B_{\Theta}^0$ is (in broad terms) defined
as follows. Consider first a pair $(\langle A, M \rangle, R(\ell) \leftarrow \rho) \in S \times \Sigma_{\Pi}$, where $(A, M)$ is an atom state, i.e., $A \subseteq J$ and $M \subseteq M_k$, and
$R(\ell) \leftarrow \rho$ is an instance of a rule in $\Pi$ over $vars(\Pi)$. Then:

1. It is the case that $\delta(\langle A, M \rangle, R(\ell) \leftarrow \rho)$ contains an atom
mapping transition of the form:

   $\langle 0, (A_1, M_1) \rangle \land \cdots \land \langle 0, (A_n, M_n) \rangle \land
   \langle 0, (j_{x_1'}, \ldots, j_{x_p'}) \rangle \land \cdots \land \langle 0, (j_{\rho}, \rho') \rangle$,

   if $A_1, \ldots, A_n$ are the children of $A$ in $J$ and the following
   conditions hold:

   (a) There is a mapping $M'$ consistent with $M$ that maps
   the atom $A$ to an atom in $\rho$. 
(b) For each $1 \leq i \leq n$, the mapping $M_i$ is precisely the restriction of $M'$ to the common variables between $A$ and $A_i$.

(c) $\text{dvar}(A) = \{j_1, \ldots, j_p\}$ and $x'_{j_i} = M'(x_{j_i})$, for each $1 \leq i \leq p$.

Intuitively, this transition maps $A$ to the current node of $T$, and then launches new copies of $B_{\Pi i}^\theta$ that check that each one of the children $A_i$ of $A$ in $J$ is matched in $T$ ($1 \leq i \leq n$). Since $J$ is a join tree, the mapping $M_i$ is the only information the automaton needs to store in order to continue consistently mapping in $T$ the subtree $J_i$ of $J$ rooted in $A_i$. This is represented by the atom $(0, (A_i, M_i))$ in the transition. In addition, if $A$ mentions a distinguished variable $x_{j_i}$, we have to ensure that it is mapped to a distinguished occurrence of $x_{j_i}'$. This is enforced with the atom $(0, (j_i, x_{j_i}))$ in the atom mapping transition, as we will see below.

2. It is the case that $\delta((A, M), R(i) \leftarrow \rho)$ contains a moving transition $(j, (A, M))$, for $j \in \{-1, 1, \ldots, \ell\}$, if one of the following condition holds:

(a) $j \in \{1, \ldots, \ell\}$ and, if $R_{i_j}(t_{i_j})$ is the $j$-th intensional atom of $\rho$, then for each variable appearing both in $A$ and the domain of $M$ it is the case that its image under $M$ is in $t_{i_j}$.

(b) $j = -1$ and for each variable appearing both in $A$ and the domain of $M$ it is the case that its image under $M$ appears in $t$.

This transition moves the automaton to an adjacent node in $T$. It is applied when the current atom $A$ cannot be mapped to the current node of $T$. The value of $M$ must be propagated through the head of the rule to which the automaton moves. This allows to ensure that $M$ satisfies the connectedness condition in the definition of strong containment mapping.

Consider now a pair $((j, x), R(i) \leftarrow \rho)$, where $(j, x)$ is a variable atom, i.e. $j \in \{1, \ldots, m\}$ and $x \in \text{vars}(\Pi)$, and $R(i) \leftarrow \rho$ is an instance of a rule of $\Pi$ over $\text{vars}(\Pi)$. Then $\delta((j, x), R(i) \leftarrow \rho)$ contains a “variable checking” transition $(-1, (j, x))$ if the variable $x$ is in $t$. In addition, it contains a variable checking transition $(0, \text{accept})$ if the $j$-th variable of $t$ is $x$. The goal of the variable checking transitions is to ensure that each distinguished occurrence of a variable $x$ in $\theta$ is mapped to a distinguished occurrence of the corresponding distinguished variable of $T$.

It can be proved that this construction is correct, that is, that $B_{\Pi i}^\theta$ defines the set of trees $T$ over $\Sigma_\Pi$ for which there is a strong containment mapping from $\theta$ to $T$. Notice that the number of states of $B_{\Pi i}^\theta$ is $O(||\theta||^{k+1}||\Pi||^{k})$, i.e., it is polynomial in the size of $\Pi$ and $\theta$. Also, for each state $s$ of $B_{\Pi i}^\theta$ and symbol $a \in \Sigma_{\Pi i}$, the size of $\delta(s, a)$ is polynomial. This finishes our sketch. 

### 4.2.1 Results for $\mathcal{AC}$ and $\text{TW}(1)$

Theorem 5 states that both problems $\text{Cont}(\text{Datalog}, \mathcal{AC})$ and $\text{Cont}(\text{Datalog}, \text{TW}(2))$ are complete for $2\text{EXPTIME}$, and that for $\text{TW}(2)$ and $\text{HW}(2)$ this holds even for schemas of fixed arity. This leaves the following two cases open: (1) $\text{Cont}(\text{Datalog}, \text{TW}(1))$, and (2) $\text{Cont}(\text{Datalog}, \mathcal{AC})$, for $c \geq 1$. We obtain that both can be solved in $\text{EXPTIME}$ as corollaries of Theorem 6.

**Corollary 1.** The following problems are complete for $\text{EXPTIME}$:

1. $\text{Cont}(\text{Datalog}, \mathcal{AC})$, for each $c \geq 1$.

2. $\text{Cont}(\text{Datalog}, \text{TW}(1))$.

**Proof (Sketch):** We concentrate on upper bounds. Consider first $\text{Cont}(\text{Datalog}, \mathcal{AC})$, for $c \geq 1$. Each input consists of a Datalog program $\Pi$ and a UCQ $\theta$ over $\sigma$, where the arity of $\sigma$ is bounded by $c$. It follows that $\theta \in \mathcal{AC}_c$, and therefore that $\Pi \subseteq \theta$ can be checked in $\text{EXPTIME}$ (from Theorem 6).

Consider now $\text{Cont}(\text{Datalog}, \text{TW}(1))$. We use the following fact, which has a simple proof: $\text{TW}(1) \subseteq \mathcal{AC}_2$. The result now follows from Theorem 6.

### 4.2.2 Results about equivalence

Checking containment of UCQ in Datalog is decidable in $\text{EXPSPACE}$ [16]. Together with Theorem 6 this implies that checking if a Datalog program is equivalent to a UCQ in $\mathcal{AC}_k$ is decidable in $\text{EXPTIME}$, for each $k \geq 1$.

**Corollary 2.** Let $k \geq 1$. The following problem is complete for $\text{EXPTIME}$: Given a Datalog program $\Pi$ and a UCQ $\theta$ in $\mathcal{AC}_k$, decide if $\Pi$ is equivalent to $\theta$.

### 4.2.3 Queries in $\mathcal{AC}_k$ modulo equivalence

Tractability results for $\text{TW}(k)$ and $\text{HW}(k)$ are invariant modulo equivalence, i.e., they continue to hold for the classes $\mathcal{H}(\text{TW}(k))$ and $\mathcal{H}(\text{HW}(k))$ of UCQs that are equivalent to one in $\text{TW}(k)$ and $\text{HW}(k)$, respectively [18, 15]. Interestingly, our single-exponential time bound for the problem $\text{Cont}(\text{Datalog}, \mathcal{AC}_k)$ is also invariant modulo equivalence for queries in $\mathcal{AC}_k$. Let $\mathcal{H}(\mathcal{AC}_k)$ be the set of all UCQs that are equivalent to one in $\mathcal{AC}_k$, for $k \geq 1$. Then:

**Proposition 3.** Let $k \geq 1$. Then $\text{Cont}(\text{Datalog}, \mathcal{H}(\mathcal{AC}_k))$ can be solved in $\text{EXPTIME}$.

**Proof:** First, we introduce the notion of strong induced subqueries. A CQ $\theta'$ is a strong induced subquery of a CQ $\theta$ if:

1. The set $V_{\theta'}$ of variables mentioned in $\theta'$ is contained in the set $V_{\theta}$ of variables mentioned in $\theta$.

2. The atoms of $\theta'$ are exactly the atoms of $\theta$ induced by the variables in $V_{\theta'}$.

3. The free variables of $\theta'$ are exactly the free variables of $\theta$.

4. If $A$ is a atom in $\theta$ but not in $\theta'$ and $V_A$ are the variables in $A$ that belong to $V_{\theta'}$, then there exists an atom $A'$ in $\theta'$ that contains all the variables in $V_A$.

As it turns out, the class $\mathcal{AC}_k$ is closed under taking strong induced subqueries. Indeed, let $\theta$ be a CQ in $\mathcal{AC}_k$ and $\theta'$ a strong induced subquery of $\theta$. Since each atom of $\theta'$ is an atom of $\theta$, it follows that no two distinct atoms in $\theta'$ share more than $k$ variables. It thus suffices to prove that $\theta'$ has a join tree.

Let $T$ be a join tree of $\theta$. It is convenient in this case to switch back and forth between two different ways of viewing at $T$: the traditional one in which each node of $T$ is an atom $A$ of $\theta$, and another one – based on the fact that $T$ can also be interpreted as a tree decomposition of $\theta$ – in which each such node is associated with the set of variables in $V_A$ that are mentioned in $A$. Consider now a node of $T$ that is associated with some atom $A$ that is not in $\theta'$. Assume that the set of variables mentioned in $A$ is $S$ (notice that $S \nsubseteq V_{\theta'}$, since $\theta'$ is a strong induced subquery of $\theta$). Let
5. PRELIMINARIES

Graph databases. Let $\Sigma$ be a finite alphabet. A graph database $\mathcal{G}$ over $\Sigma$ is a pair $(V, E)$, where $V$ is a finite set of nodes and $E \subseteq V \times \Sigma \times V$. Thus, each edge in $\mathcal{G}$ is a triple $(v, a, v') \in V \times \Sigma \times V$, whose interpretation is an $\alpha$-labeled edge from $v$ to $v'$ in $\mathcal{G}$. A path in a graph database $\mathcal{G} = (V, E)$ is a sequence

$$\eta = v_0 \alpha_0 v_1 \alpha_1 v_2 \cdots v_k \alpha_k v_{k+1},$$

for $k \geq 0$, such that $((v_{i-1}, a_{i-1}, v_i))$ is in $E$, for each $i$ with $1 \leq i \leq k$. The label of $\eta$, denoted $\lambda(\eta)$, is the string $a_0 a_1 \cdots a_{k-1}$. Notice that $\eta$ is a path, for each $v \in V$. The label of such path is the empty string $\varepsilon$.

C2RPQs and Datalog. Queries over graph databases are typically navigational, in the sense that they allow to recursively traverse the edges of the graph while checking for the existence of paths satisfying a regular condition (see, e.g., [1, 35]).

The basic mechanism for querying graph databases is the class of two-way regular path queries, or 2RPQs [9]. A 2RPQ $L$ over $\Sigma$ is a regular expression over the alphabet $\Sigma^*$ that extends $\Sigma$ with the inverse $\cdot^*$ of each symbol $a \in \Sigma$. To define the semantics of 2RPQs we use the notion of the completion of a graph database $\mathcal{G}$, denoted $\mathcal{G}^\times$. This is the graph database over $\Sigma^*$ that is obtained from $\mathcal{G} = (V, E)$ by adding the edge $(u, a^* v)$, for each $(v, a, u) \in E$. We define the evaluation $L(\mathcal{G})$ of a 2RPQ $L$ over $\mathcal{G}$ to be the set of pairs $(v, v')$ of nodes in $V$ such that there is a path $\eta$ in $\mathcal{G}^\times$ from $v$ to $v'$ whose label $\lambda(\eta)$ satisfies $L$.

The analogue of CQs in the context of graph databases is the class of conjunctive 2RPQs, or C2RPQs [7], that closes 2RPQs under joins and projection. Formally, a C2RPQ $\gamma$ over $\Sigma$ is an expression of the form

$$\exists x_1 \cdots \exists x_n. (L_1(x_1, y_1) \wedge \cdots \wedge L_m(x_m, y_m)),$$

where each $L_i$ is a 2RPQ over $\Sigma$, for $1 \leq i \leq m$, the $x_i$’s and $y_i$’s are variables, and $\gamma$ is a tuple of variables among the $x_i$’s and $y_i$’s. Again, $\gamma(\bar{x})$ denotes that $\bar{x}$ is the tuple of free variables of $\gamma$.

A homomorphism from a C2RPQ $\gamma = \exists x_1 \cdots \exists x_n. (L_1(x_1, y_1) \wedge \cdots \wedge L_m(x_m, y_m))$ to a graph database $\mathcal{G} = (V, E)$ is a mapping $h$ from the set of variables used in $\gamma$ to $V$, such that $(h(x_i), h(y_i)) \in L_i(\mathcal{G})$, for each $1 \leq i \leq m$. The evaluation $\gamma(\mathcal{G})$ of $\gamma$ over $\mathcal{G}$ is the set of tuples $h(\bar{x})$, for $h$ a homomorphism from $\gamma$ to $\mathcal{G}$.

A UC2RPQ $\Gamma$ is a finite set $(\gamma_1(\bar{x}), \ldots, \gamma_k(\bar{x}))$ of C2RPQs over $\Sigma$ with the same free variables. In addition, we define $\Gamma(\mathcal{G})$ to be $\bigcup_{1 \leq i \leq k} \gamma_i(\mathcal{G})$, for each graph database $\mathcal{G}$.

It is clear that each graph database $\mathcal{G} = (V, E)$ over $\Sigma$ can be represented as a relational database $\mathcal{D}(\mathcal{G})$ over the schema $\sigma(\Sigma)$ that consists of one binary relation symbol $E_a$, for each symbol $a \in \Sigma$. The database $\mathcal{D}(\mathcal{G})$ consists of all facts of the form $E_a(v, v')$ such that $(v, a, v') \in E$ (where $(v, v') \in V$ and $a \in \Sigma$). We identify Datalog programs $\Pi$ over $\Sigma$ with Datalog programs over $\sigma(\Sigma)$, and define $\Pi(\mathcal{G})$ to be $\Pi(\mathcal{D}(\mathcal{G}))$, for each graph database $\mathcal{G}$. Notice that while each symbol in $\sigma(\Sigma)$ is binary, the intensional symbols used in $\Pi$ can be of arbitrary arity. It is a well-known fact that Datalog subsumes UC2RPQs over graph databases.

Containment of Datalog in UC2RPQs. In this section we study the containment problem of Datalog in classes $\mathcal{C}$ of UC2RPQs, which (by slightly abusing notation) we denote by $\text{Cont}(\text{Datalog}, \mathcal{C})$. Formally, $\text{Cont}(\text{Datalog}, \mathcal{C})$ is defined as follows: Given a Datalog program $\Pi$ and a UC2RPQ $\Gamma \in \mathcal{C}$ over the same finite alphabet $\Sigma$, is it the case that $\Pi \subseteq \Gamma$ (that is, $\Pi(\mathcal{G}) \subseteq \Gamma(\mathcal{G})$ for every graph database $\mathcal{G}$)?
Calvanese et al. proved that \text{Cont}(\text{Datalog,UC2RPQ}) is in general not only decidable, but also not more expensive than the problem \text{Cont}(\text{Datalog,UCQ}).

**Theorem 7.** [8] The problem \text{Cont}(\text{Datalog,UC2RPQ}) is complete for \text{2EXPTIME}.

### 5.2 Containment of Datalog in Acyclic UC2RPQs

In general, evaluation of UC2PQs is NP-complete [1], but \text{acyclic} UC2RPQs are tractable. To restrict the syntactic shape of queries to be \text{acyclic} [3].

This notion is typically defined in terms of the acyclicity of its underlying conjunctive query. Let \( \gamma = \exists y \bigwedge_{1 \leq i \leq m} L_i(x, y_i) \) be a C2RPQ. Its underlying CQ is the query over the schema of binary relation symbols \( T_1, \ldots, T_m \) defined as: \( \exists y \bigwedge_{1 \leq i \leq m} T_i(x, y_i) \). Intuitively, this underlying conjunctive query represents the structure of \( \gamma \) when the regular languages that label the atoms of \( \gamma \) are turned into relation symbols.

A C2RPQ is \text{acyclic} if its underlying CQ is acyclic (or, equivalently, if it belongs to \text{TW}(1), since \( \mathcal{A}_{\text{C}} = \text{TW}(1) \) for binary schemas).

**Example 5.** Let \( L_1, L_2 \) and \( L_3 \) be regular expressions over \( \Sigma \). The C2RPQ \( L_1(x, x) \land L_2(x, y) \land L_3(y, x) \) is \text{acyclic}. The C2RPQ \( L_1(x, y) \land L_2(y, z) \land L_3(z, x) \) is not \text{acyclic}.

We proved in Theorem 6 that containment of Datalog in UCQs in \text{TW}(1) is in \text{EXPTIME}. Since acyclic UC2RPQs are those that contain only C2RPQs whose underlying CQ is of treewidth one, it is natural to study whether the restriction to \text{acyclic} UC2RPQs also helps alleviate the complexity of \text{Cont}(\text{Datalog,UC2RPQs}). We prove that this is not the case, by showing that the 2EXPTIME-hardness proof for \text{Cont}(\text{Datalog,UC2RPQ}) in [8] can be carried out even with UC2RPQs in \text{ACR}.

**Theorem 8.** The problem \text{Cont}(\text{Datalog,ACR}) is complete for \text{2EXPTIME}.

### 5.3 Better bounds for queries in \text{ACR}^k

In the case of \text{Cont}(\text{Datalog,ACR}), a big source of complexity was the presence of pairs of atoms in queries in \text{ACR} that share an unbounded number of variables. In the case of \text{Cont}(\text{Datalog,ACR}) we identify a different source of complexity, which is the existence of queries in \text{ACR} with an unbounded number of atoms connecting the same variables. By restricting this parameter we again obtain a hierarchy inside the class of \text{acyclic} queries, which in this case is defined by queries in \text{ACR} with a bounded number of atoms connecting pairs of distinct variables. We prove that containment of \text{Datalog} in any of the levels of this hierarchy is in \text{EXPTIME} (and it is actually complete for this class).

Formally, let us define \text{ACR}^k, for \( k \geq 1 \), to be the class of \text{queries} \( \Gamma \) in \text{ACR} such that for each C2RPQ \( \gamma \in \Gamma \) and pair \((x, x')\) of distinct variables in \( \gamma \), there are at most \( k \) atoms of the form \( L(x, x') \) or \( L(x', x) \) in \( \gamma \). Clearly:

\[
\text{ACR}^1 \subset \text{ACR}^2 \subset \cdots \subset \text{ACR}^k \subset \cdots \subset \text{ACR}.
\]

**Example 6.** The C2RPQ \( L_1(x, x) \land L_2(x, y) \land L_3(y, x) \) is in \text{ACR}^2.

We prove next that \text{Cont}(\text{Datalog,ACR}^k) can be solved in single-exponential time, for each \( k \geq 1 \).

**Theorem 9.** Let \( k \geq 1 \). Then \text{Cont}(\text{Datalog,ACR}^k) is complete for \text{EXPTIME}.

**Proof (Sketch):** Containment of a Datalog program \( \Pi \) in a C2RPQ \( \Gamma \) can be stated in terms of the existence of \text{strong containment mappings} from the \text{expansions} of the C2RPQs in \( \Gamma \) to the proof trees of \( \Pi \). An expansion of a C2RPQ \( \gamma = \exists y \bigwedge_{1 \leq i \leq m} L_i(x, y_i) \) over \( \Sigma \) is a CQ over \( \sigma(\Sigma^\gamma) \) of the form \( \phi = \exists y \bigwedge_{1 \leq i \leq m} \theta_i(x, y_i) \), where for each \( i \) with \( 1 \leq i \leq m \) it is the case that \( \theta_i(x, y_i) \) is the form:

\[
\exists u_1 \cdots u_{n-1} (E_{a_1}(u_1, u_1) \land E_{a_2}(u_1, u_2) \cdots \land E_{a_n}(u_{n-1}, y_i)),
\]

for \( a_1, a_2, \ldots, a_n \) a word over \( \Sigma^\gamma \) that satisfies \( L_1 \). We assume no two distinct \( \theta_i \)'s share an existentially quantified variable. Strong containment mappings \( \mu \) from the expansion \( \phi \) to the proof tree \( \nu \) of \( \Pi \) are defined as in the proof of Theorem 6, save for condition (iii) that is restated as follows: for each atom \( A \) in \( \theta \) we have that if \( A \) is of the form \( E_{a}(x, y), \) for \( a \in \Sigma \), then \( \mu_a(\mu(x), \mu(y)) \) appears in \( \nu \), and if \( A \) is of the form \( E_{a}(x, y), \) for \( a \in \Sigma \), then \( \mu_a(\mu(x), \mu(y)) \) appears in \( \nu \).

For each pair \((x, x')\) of variables in \( \gamma \), we denote by \( \gamma(x, x') \) the set of atoms of the form \( L(x, x') \) or \( L(x', x) \) in \( \gamma \). Consider the undirected graph \( G_\gamma \), whose vertices are the variables of \( \gamma \), and there is an undirected edge between \( x \) and \( x' \) in \( G_\gamma \) if \( \gamma(x, x') \) is nonempty. Since \( \gamma \) is \text{acyclic}, the graph \( G_\gamma \) is a tree. We assume for the sake of this sketch that \( G_\gamma \) is connected and contains no loops, but both cases can be handled at the cost of a more cumbersome construction.

The \( 2\text{ATA} B_\gamma^1 \) reads \( G_\gamma \) in a top-down fashion, looking for the existence of an expansion \( \phi \) of \( \gamma \) and a strong containment mapping \( \mu \) from \( \phi \) to \( \nu \). After scanning node \( x \) in \( G_\gamma \), the automaton stores the value of \( \mu(x) \), and then makes a universal transition to check that for each child \( y \) of \( x \) in \( G_\gamma \), the mapping \( \mu \) can be extended to a strong containment mapping from some expansion of the atoms in \( \gamma(x, y) \) to \( \nu \).

Suppose first that \( \gamma \) belongs to \text{ACR}^1. Then there is exactly one atom in \( \gamma(x, y) \), for each child \( y \) of \( x \) in \( G_\gamma \). Assume without loss of generality that \( \gamma(x, y) = \{L(x, y)\} \). Thus, in order to check that \( \mu \) can be extended to a strong containment mapping \( \mu \gamma \) from an expansion of \( L(x, y) \) to \( \nu \), the automaton \( B_\gamma^1 \) proceeds as follows: (1) it first guesses an expansion \( E_{a_1}(u_1, u_1) \land \cdots \land E_{a_n}(u_{n-1}, u_n) \) of the C2RPQ \( L(x, y) \), where \( u_0 = x \) and \( u_n = y \), and then (2) checks, iteratively from \( i = 1 \) to \( n \), that \( E_{a_1}(u_{i-1}, u_i) \) can be mapped to \( \nu \) if \( 1 \leq i \leq n \), the invariant in each iteration is that \( \mu_1(u_{i-1}) \) is already known and the automaton looks for a value \( \mu_1(u_i) \) that satisfies \( E_{a_1}(u_{i-1}, u_i) \). If such a value does not exist in the rule \( \gamma \), where \( \gamma \) labels the node in \( \nu \) currently being scanned by the automaton, then it moves to an adjacent node while checking that \( \mu_1(u_{i-1}) \) can be propagated through the head of the rule to which the automaton moves. This ensures that the two occurrences
of $u \cdots u_1$ in $E_{1 \cdots 1} (u_{i-2}, u_{i-1})$ and $E_{u_1} (u_{i-1}, u_i)$ are actually connected.

Suppose now that $\gamma$ belongs to ACR$^\gamma$, for $k \geq 2$, and, in particular, that $\gamma(x, y)$ contains more than just a single atom. Assume for the sake of the argument that $\gamma(x, y)$ consists only of two different atoms, $L_1(x, y)$ and $L_2(x, y)$. This case is more difficult than the previous one, since we now have to check that $\mu_{L_1}(y)$ and $\mu_{L_2}(y)$ are connected occurrences of the same variable in $\nu$. We explain next how to do this without incurring in an exponential blowup in the construction of $B^\gamma_{ii}$ (recall that $B^\gamma_{ii}$ must be of polynomial size).

The 2ATA $B^\gamma_{ii}$ processes the atoms $L_1(x, y)$ and $L_2(x, y)$ simultaneously; this allows to check that $\mu_{L_1}(y)$ and $\mu_{L_2}(y)$ correspond to connected occurrences of the same variable in $\nu$. This idea is implemented in $B^\gamma_{ii}$ using multideg states and transitions, which interact with atom states. We only provide an idea of how these states and transitions are used, since the actual construction of $B^\gamma_{ii}$ is intricate (and left to the full version of the paper).

We assume that $L_1$ and $L_2$ are given as nondeterministic finite automata (NFA). For states $s$ and $s'$ in $L_1$, we denote by $(L_1)_s$ the NFA that is obtained from $L_1$ by setting $s$ to be the initial state, and by $(L_1)_{s, s'}$ the NFA obtained from $L_1$ by setting $s$ and $s'$ to be the initial and final state, respectively. Similarly for $L_2$. For $i \in \{1, 2\}$, the atom states associated with $L_i$ in $B^\gamma_{ii}$ are of the form $(L_i)_{s, \gamma}(u, v)$ or $(L_i)_{s, \gamma}(u, v)$, where $s$ and $s'$ are states in $L_i$, and $u$ and $v$ are variables mentioned in $\nu$ (i.e., $u, v \in \text{vars}(\Pi)$). When the automaton is in state $(L_i)_{s, \gamma}(u, v)$ (the case $(L_i)_{s, \gamma}(u, v)$ is analogous), the automaton $\nu$ in $\gamma(x, y)$ is in state $(\gamma(x, y))_u$ (the case $(\gamma(x, y))_v$ is analogous), and $u$ and $v$ are variables in the rule $r$ that labels the node currently being scanned by $B^\gamma_{ii}$. The 2ATA $B^\gamma_{ii}$ then looks for an expansion $\theta = \exists u_1 \cdots \exists u_{i-1} (E_{u_1}(x', u_1) \land \cdots \land E_{u_{i-1}}(x', u_{i-1}))$ of the 2CRQP $(\gamma(x, y))'_u$, $x'$ and $y'$ are fresh variables, and a strong containment mapping $\mu_0 \theta$ from $\theta$ to $\nu$ such that $\mu_0(x')$ corresponds to an occurrence of $u$ in $\nu$ that is connected to the occurrence of $u$ in $\gamma(x, y)$ and similarly for $\mu_0(y')$ and $\nu$. Although the implementation of atom states requires some work, we can adapt techniques from [8] in order to construct the modules of $B^\gamma_{ii}$ that implement them.

Now we explain how to process the atoms $L_1(x, y)$ and $L_2(x, y)$ in $\gamma(x, y)$ simultaneously. The multideg states associated with $\gamma(x, y)$ in $B^\gamma_{ii}$ are of the form $\gamma_{x, y}(s_1, s_2; u_1, u_2)$, where $s_1$ and $s_2$ are states of $L_1$ and $L_2$, respectively, and $u_1$ and $u_2$ are variables mentioned in $\nu$. When processing $\gamma(x, y)$, the automaton looks for expansions $\theta_1$ and $\theta_2$ of $(L_1(s, \gamma), x, y)$, respectively, and a strong containment mapping $\mu_{L_1, L_2}$ from $\theta_1$ and $\theta_2$ to $\nu$, respectively, such that (1) $\mu_{L_1, L_2}(z_1)$ is an occurrence of $u_1$ that is connected to the occurrence of $u_1$ in $\nu$, and similarly for $\mu_{L_2, L_2}(z_2)$ and $u_2$, and (2) $\mu_{L_2, L_2}(y)$ and $\mu_{L_2, L_2}(y)$ are connected occurrences of the same variable in $\nu$. We explain next how this is done.

We consider two cases:

1. The rule $r$ currently being scanned contains an occurrence of $w$ that is connected to $\mu_{L_1}(y)$ and $\mu_{L_2}(y)$. This case is simple since we can continue processing $\theta_1$ and $\theta_2$ independently from the current node. This is done by performing a universal transition $(0, (L_1)_{s, \gamma}(u_1, w)) \wedge (0, (L_2)_{s, \gamma}(u_2, w))$.

2. Otherwise, it is easy to prove that there is $j \in \{-1, 1, \ldots, \ell\}$, where $\ell$ is the number of intensional predicates in the rule $r$, such that both $\mu_{L_1}(y)$ and $\mu_{L_2}(y)$ belong to the subtree $v_j$ of $\nu$ rooted at the $j$-th child of the node currently being scanned, if $j \geq 1$, and to the subtree $v_{j-1}$ of $\nu$ induced by all descendants of the proper ancestors of the current node, if $j = -1$.

One might be tempted to apply a “moving” transition (see the proof of Theorem 6) in direction $j$, and check that $u_1$ and $u_2$ are propagated through the head of the rule in such direction. Nevertheless, it could be the case that $v_j$ does not contain any occurrence of a variable connected to $u_1$ or $u_2$. Indeed, some “prefixes” of the expansions $\theta_1$ and $\theta_2$ could be mapped outside $v_j$. Therefore, before moving in direction $j$, we have to update the state $\gamma_{x, y}(s_1, s_2; u_1, u_2)$. Intuitively, the automaton $B^\gamma_{ii}$ guesses “prefixes” of the expansions $\theta_1$ and $\theta_2$ that are mapped outside $v_j$ via $\mu_{L_1}$ and $\mu_{L_2}$, and continues processing the corresponding “suffixes”.

More formally, let $A$ be the first atom in the expansion $\theta_1$ that is mapped via $\mu_{L_1}$ inside $v_j$. If $A$ is the first atom in the expansion $\theta_1$, then we let $u_1'$ to be $u_1$. Otherwise, let $A'$ be the atom that precedes $A$ in $\theta_1$ (notice that $A'$ must be mapped via $\mu_{L_1}$ outside $v_j$). In this case, we let $u_1' = u_1$ be the one variable $A$ and $A'$ have in common. Analogously, we define $u_2'$ with respect to $\theta_2$ and $\mu_{L_2}$. Notice that rule $r$ must contain occurrences of $u_1'$ and $u_2'$, and $\mu_{L_1}$, $\mu_{L_2}$, and $\mu_{L_2}$, and continues processing the corresponding “suffixes”.

We can adapt techniques from [8] to guess the “prefixes” of the expansions that will be processed in future transitions.

Note that case (1) can only apply a finite number of times. After this case (2) applies, and the expansions $\theta_1$ and $\theta_2$ of $(L_1(x, y)$ and $L_2(x, y)$ have been correctly mapped to $\nu$.

In general, the number of multideg states is $O(||\gamma||^{k+1}||\Pi||^k)$, and the number of atoms states is $O(||\gamma||^2||\Pi||^2)$. Therefore, the number of states in $B^\gamma_{ii}$ is polynomial in the size of $\Pi$ and $\gamma$. In addition, there is a polynomial that bounds the size of $\delta(t, a)$, for each state $t$ of $B^\gamma_{ii}$ and symbol $a \in \Sigma_1$. This finishes the sketch of the proof.

Notice that containment in the opposite direction, that is, checking if a query in ACR is contained in a Datalog program, is undecidable [8] (same for equivalence).

5.3.1 Queries of larger treewidth

We prove here that it is not possible to extend this positive result to classes of UC2RPQs of larger treewidth. Formally, let TWR(2) be the class of UC2RPQs $\Gamma$ such that for each C2RPQ $\gamma \in \Gamma$ we
have that (1) the underlying CQ of $\gamma$ is of treewidth at most two, and (2) there is at most one atom of the form $L(x, x')$ or $L(x', x)$ in $\gamma$ for each pair $(x, x')$ of distinct variables. Then:

**Proposition 5.** The problem $\text{Cont}($Datalog,$\text{TWR}(2))$ is complete for 2EXPTIME.

6. CONCLUSIONS

Our results convey two messages: (1) Traditional restrictions on UCQs and UC2RPQs that have been used for reducing the complexity of the UCQ and UC2RPQ evaluation problem are not adequate for reducing the complexity of $\text{Cont}($Datalog,$\text{UC})$ and $\text{Cont}($Datalog,$\text{UC2RPQ})$. (2) Adequate restrictions can be identified by unveiling new hierarchies of acyclic UCQs and UC2RPQs.

We are currently investigating what is the impact of applying the traditional notion of bounded (hyper-)treewidth in restricted versions of our problem. For instance, it has recently been proved that checking containment of the Datalog program II in the UCQ $\Theta$ is 2EXPTIME-complete even if II is monadic, i.e., it contains only monadic intensional symbols [5]. It is by no means clear whether better complexity bounds for this problem can be obtained by restricting $\Theta$ to be in TW$(k)$ or HW$(k)$, for $k \geq 1$.

Acknowledgments Barceló is funded by Fondecyt grant 1130104. Romero is funded by CONICYT Ph.D. Scholarship.

7. REFERENCES